Delphi e RIA: Interfaces ricas para WEB, com Delphi

Clube Delphi

cxGrid do DevExpress
Aprenda como enriquecer o visual da VCL

Ano 13 :: Edição 147 :: R$ 11,90

DevMedia

Delphi XE 3

O que há de novidade

Análise de código
Métricas para qualidade e para estimativas de tempo

ORM e Delphi
Utilizando o framework OpenSource DORM no Delphi

ISSN 1517990-7

TENHA ACESSO ILIMITADO A TODO CONTEÚDO DO SITE DEVMEDIA

SEJA UM ASSINANTE MVP!

Tenha acesso garantido a qualquer tema que você possa necessitar!

VIDEOS -AULAS

O assinante MVP acessa todas as vídeo-aulas do site DevMedia. Nenhum outro produto dá essa possibilidade. São mais de **7.000 video-aulas.**

CURSOS

Nosso site possui atualmente **+ de 250 cursos online** com a qualidade que você já conhece e confia. Com o MVP você terá acesso a todos eles.

REVISTAS

Assinante MVP acessa todas as nossas revistas na versão digital, incluindo todas as edições anteriores. Quando pintar aquela dúvida você terá mais de **3.500 artigos para consultar!**

MVP
MOST VALUABLE PROFESSIONAL

A assinatura MVP te dá a oportunidade de aumentar seu conhecimento em diversos assuntos, com qualidade e economia.

Invista na sua carreira se torne um profissional ainda mais completo!

POR APENAS R$59,90 POR MÊS

DevMedia

Acesse: www.devmedia.com.br/mvp

Sumário

Delphi

Delphi/ Novidades — **05 – Delphi XE3** [*Fabrício Hissao Kawata*]

Minicurso — **13 – RIA com Delphi – Parte 1** [*Giuliano Scombatti Pinto*]

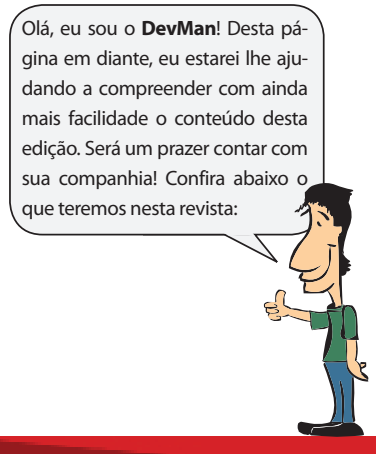
Delphi/ Easy — **20 – Componente cxGrid da DevExpress** [*Sander Carlos Mendes*]

Delphi/ Novidades — **30 – Metropolis Ui Applications** [*Fabrício Hissao Kawata*]

Boas Práticas

Boas Práticas — **38 – DORM** [*Giuliano Scombatti Pinto*]

Engenharia de Software — **44 – Análise de código** [*Paulo Quicoli*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Clube Delphi

Ano 13 - 147ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@gmail.com

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Guinther Pauli
guintherpauli@gmail.com

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Giuliano Scombatti Pinto
giuliano@sygnux.com.br

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

A Web mudou! Desde o seu surgimento até agora, diversos acontecimentos vem marcando essa mudança. Em um breve retrospecto, podemos lembrar a época em que o desenvolvimento de páginas e aplicações para Web era baseado em animações por meio de Flash, possuindo até hoje seus adeptos. O grande problema neste caso baseava-se nas dificuldades de utilização destes websites pelos mecanismos de busca, porém, com RIA tudo isso mudou! As aplicações de interface rica conquistaram (e vem conquistando) cada vez mais seu espaço. A primeira impressão é impactante quando vemos o resultado de aplicações criadas em Silverlight e Adobe Flex por exemplo. O suporte a este tipo de aplicação foi incluído por meio do Prism, no entanto, veremos que é possível utilizá-lo através do próprio Delphi. Nessa edição o Giuliano inicia um minicurso que objetiva exatamente demonstrar como aplicar RIA em suas aplicações por meio do Framework UniGui. Ele ainda está na sessão boas práticas, onde faz uma abordagem de um tema que agrada muitos desenvolvedores, a persistência de dados. Veremos como utilizar o DORM para realizar esta tarefa em bancos de dados relacionais.

O Fabrício nos dá um Overview sobre o Delphi XE3, abordando suas principais novidades como o Metropolis, uma abordagem sobre o reformulado mecanismo de Binding de objetos, Firemonkey e uma breve explicação sobre o que há de novo no Prism XE3 e HTM5 Builder, a mais nova ferramenta da Embarcadero que substituí até então a solução RAD PHP.

Na sessão Easy o Sander Mender apresenta uma das mais renomadas suítes de componentes para Delphi do mercado, a DevExpress, ilustrando em um exemplo prático como usar e abusar do componente cxGrid, extraindo o que há de melhor de seus recursos de uma maneira customizável e simples.

E para finalizar, eu abordo algumas questões que estão relacionadas com engenharia de software. Apresento alguma métricas que podem dizer se seu código é de qualidade ou não, e também apresento outro tipo de métrica, que lhe ajudará a dizer, por exemplo, quanto tempo você pode levar para concluir uma série de funcionalidades, uma vez que você descobrir o tamanho do projeto em pontos de função. Boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DevMedia

Delphi XE3

Um overview das novidades da mais recente versão do Delphi

Assim como sua versão anterior (XE2), o recém-lançado Delphi XE3 é fundamentado na capacidade de se construir aplicações de forma totalmente RAD, para as plataformas Windows e Mac. Contudo, como adicional nesta versão, está a inclusão de melhorias e novos recursos que contemplam o desenvolvimento de aplicativos tradicionais Windows, passando por Mac OS X Mountain Lion e culminando no novo Windows 8. Este último, aliás, promete agregar uma nova geração de aplicações, que serão então executadas tanto em Desktops quanto em Tablets, incluindo uma interface de usuário muito particular.

Em relação ao Windows 8, o Delphi XE3 provê uma série de modelos (Templates) de projetos pré-prontos e exclusivos para a plataforma, que incluem características que vão da aparência (layout) ao suporte a toque (touch). Tais modelos abrangem então um novo recurso denominado Metropolis UI, que proporciona a similaridade e compatibilidade com os modelos de aplicativos mais populares do Windows 8. A fim de expandir este novo conceito visual e de experiência interativa, há ainda a possibilidade de se converter os formulários de aplicações tradicionais VCL ou FMX, num processo totalmente simples e intuitivo, como já é de praxe na grande maioria das operações feitas no Delphi.

Ainda no XE3 a plataforma de aplicações FireMonkey continua sua evolução e apresenta sua nova versão, denominada FM2. As mudanças e melhorias, neste caso, se fazem presente com relação à capacidade e desempenho das aplicações. Prova disso é a presença de novos componentes disponíveis para uso, tal como um novo controle de Grid e ActionLists (similar ao da VCL). Adicionalmente, da mesma forma que o FireMonkey, o LiveBindings é outro recurso introduzido no XE2 e aprimorado nesta versão. Isto porque, agora a "ligação de dados" pode ser feita de maneira totalmente visual, sem a necessidade de escrita de código, como era anteriormente. Para que isso se torne possível, a ferramenta agora provê um novo recurso denominado Visual LiveBindings, que possibilita, por exemplo, o estabelecimento de um "Binding" entre um controle UI e um DataSet através de uma simples ação de arrastar-e-soltar (Drag-and-Drop). Em vista do que foi mencionado até aqui, podemos determinar que três itens se destacam como sendo as principais novidades desta versão XE3 do Delphi: Metropolis UI, FireMonkey FM2 e Visual LiveBindings. Obviamente que outras novidades e diversas melhorias mais sutis complementam este Release do produto. Um mix de tudo isso é o que será abordado a seguir, fazendo com que o leitor/desenvolvedor se sinta situado com relação ao estágio atual de sua ferramenta de trabalho.

Resumo DevMan

Delphi XE3:

Metropolis UI, FireMonkey FM2 e Visual LiveBindings são as mais recentes terminologias incorporadas ao mundo Delphi nesta versão XE3 do produto. Versão esta que acaba de ser lançada no mercado, num instante próximo ao nascimento de outra importante figura tecnológica, o aclamado Windows 8, que vem sob o status de revolucionar os cenários de Desktops e Tablets abrangidos pelo contexto Microsoft. Sendo assim, o Delphi mais uma vez se faz presente, provendo novos recursos a seus utilizadores a fim de proporcionar mais opções de desenvolvimento, acompanhando as tendências de mercado. Conforme será visto, prova disso são os novos tipos de projetos voltados à construção de aplicações ricas com suporte a toque e gestos, já direcionadas ao Windows 8. Dito isto, a intenção do artigo é a de fazer um Overview sobre as novidades do Delphi XE3, expondo as nuances de cada novo recurso e onde estes poderão ser empregados. Com o intuito de contribuir para o aprendizado, além da abordagem teórica conceitual, sempre que possível, serão expostos exemplos práticos do elemento em questão.

Em que situação o tema é útil:

O tema é útil para demonstrar a evolução tecnológica do mercado em relação ao Delphi, tal qual o advento das plataformas Mac OS X Mountain Lion e do aclamado Windows 8, a versão XE3 apura ainda mais sua linhagem, provendo recursos para a construção de aplicações para estes novos cenários. Portanto, através do Delphi XE3 a comunidade poderá estar de acordo com o que há de mais novo no mercado de Software e tecnologia.

Metropolis UI

Metropolis UI pode ser rapidamente definido como sendo um novo agregado de recursos e características, cuja função princi-

pal é a de reproduzir aplicações Delphi compatíveis com o novo modelo de UI (User Interface) trazida pelo Windows 8. De início, podemos cravar que aplicações que vislumbram seguir este modelo deverão ser habilitadas a trabalhar tanto em contextos tradicionais, traduzidos no uso do mouse, como também em contextos sensíveis ao toque. Adicionalmente, poderão ainda ser executadas tanto em Desktop, como em dispositivos móveis. Tal modelo, inicialmente batizado como Metro pela Microsoft, prima, entre outras coisas, por um visual mais limpo, com uso de blocos informativos chamados de Tiles, além do suporte aos recursos de toque e gestos. Dito isto, o Delphi XE3 então adere a esta tendência, provendo novos modelos de projetos (**Figura 1**) para suas duas plataformas de aplicação – VCL e FireMonkey.

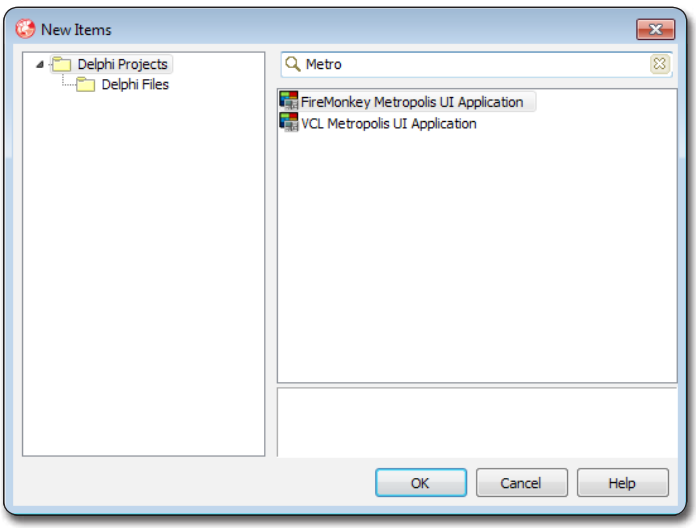


Figura 1. Novos templates de projetos Metropolis UI

Vale ressaltar ainda que a aplicação produzida por estes novos modelos de projetos, podem ser devidamente executadas nas demais versões do Windows, não se restringindo somente à versão 8. Isto se justifica pelo fato de um projeto Metropolis UI, tal como o tipo VCL Metropolis UI Application, nada mais é do que um projeto VCL tradicional, contendo elementos que o habilitam a se tornar compatível ao novo modelo de UI do Windows 8. Todos estes detalhes podem ser vistos mais de perto, no momento efetivo da criação de uma nova aplicação deste tipo. Tomando então como base a **Figura 2**, que mostra o formulário padrão, ainda em modo Design, de um novo projeto VCL Metropolis UI, podemos notar a presença de controles e componentes comuns ao desenvolvimento VCL tradicional. Pelo simples passar do mouse sobre os elementos, nota-se a presença de itens como GestureManager, ActionList, Image, Panel, Label, além de outros, e que torna o trabalho de desenvolvimento totalmente familiar e facilitado. Um comentário importante a se fazer neste momento é que apesar das novidades relacionadas a Windows 8, o XE3 ainda não provê suporte a aplicações nativas WinRT (ver **nota do DevMan 1**), bem como aplicações móveis Metropolis.

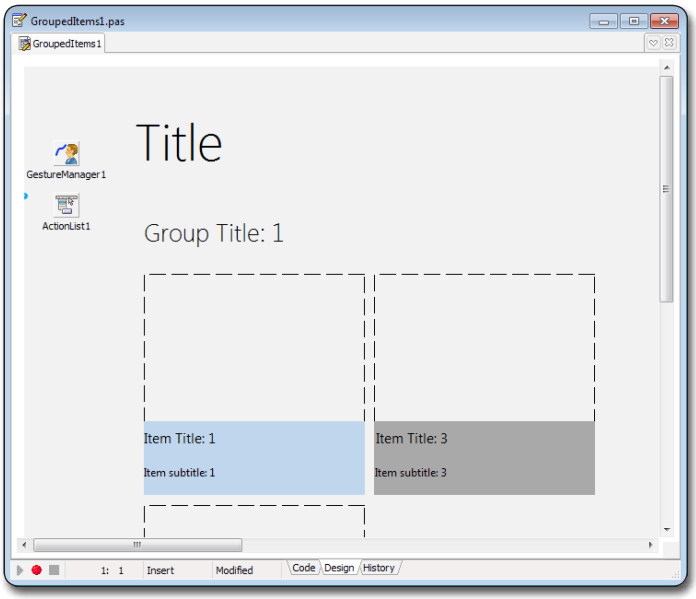


Figura 2. Form em modo Design de uma aplicação VCL Metropolis UI

Nota do DevMan 1

Windows Runtime, ou simplesmente WinRT, é uma nova e nativa arquitetura de aplicação da Microsoft e introduzida no novo Sistema Operacional Windows 8. Surge como uma API alternativa a tradicional Win32, essencialmente baseado em COM e implementando num ambiente totalmente distinto de execução. Esta nova API é escrita na linguagem C++, já projetada com melhorias de recursos e orientação a objetos, tal como o suporte a reflexão por parte de todos os seus objetos. Apesar da novidade, o WinRT não surge como um substituto imediato ao Win32, sendo ambas mantidas em paralelo para atender cenários distintos.

Uma vez acionados, podemos perceber que estes novos modelos de projetos dão início a um Wizard. Isto porque, o Delphi provê nativamente três tipos diferentes de Layout para estas aplicações. Sendo assim, no início do processo, no momento da seleção de um modelo Metropolis UI, é mostrada uma caixa de diálogo contendo as opções disponíveis. Como exemplo, a **Figura 3** mostra as opções para uma aplicação FireMonkey Metropolis UI Application que, por sinal, são as mesmas disponíveis para VCL Metropolis UI Application. A própria nomeação de cada tipo deste reflete sua característica, sendo que a principal diferença entre uma ou outra está relacionada principalmente à parte visual e de navegação do usuário. A seguir, é feita uma breve explanação sobre cada um:

- Blank Metropolis UI Application: cria uma aplicação com Layout em branco, ou seja, sem um layout definido, deixando a cargo do desenvolvedor sua criação e estilização. Apesar disso, em termos de usabilidade, ela dispõe dos recursos necessários para este tipo de UI, tais como suporte a Touch e Gestures. Em Runtime, o formulário da aplicação assume um estilo padrão, denominado MetroDark;
- Grid Metropolis UI Application: cria uma aplicação, cujo Layout apresenta uma série de grupos de itens informativos dispostos

em colunas, tal como em um Grid. Cada grupo possui então uma página de detalhes associada, acionada em Runtime no momento do clique sobre cada um. Estruturalmente falando, ainda na sua criação, são adicionados automaticamente dois formulários ao projeto. O primeiro exibindo os grupos de itens e o segundo a área de detalhes;

• Split Pane Metropolis UI Application: cria uma aplicação com Layout muito próximo do tipo anterior, com a diferença de apresentar menos itens em tela. Em Runtime, o clique sobre cada grupo leva a uma página dividida, contendo uma lista de itens à esquerda e uma seção de detalhes à direita. Por este fato, em termos de estrutura, novamente dois formulários são automaticamente criados junto do projeto.

Como exemplo do atrativo visual proporcionado pelas aplicações Metropolis, as **Figuras 4** e **5** exibem uma VCL Metropolis UI Application que usa o tipo de layout Grid.

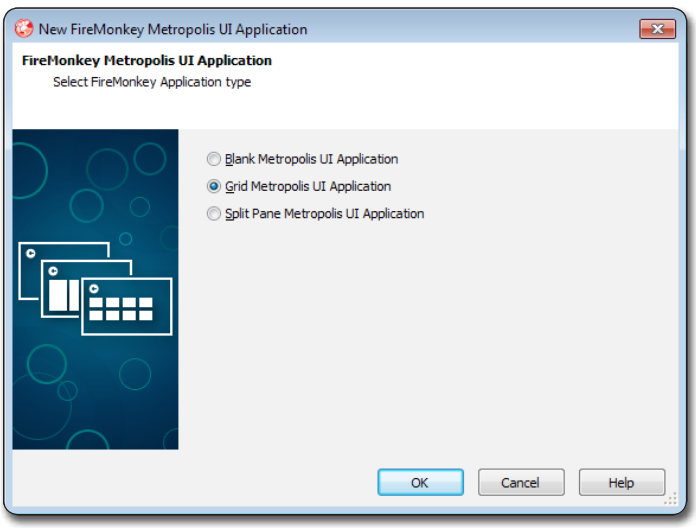


Figura 3. Selecionando o tipo de layout da aplicação FireMonkey Metropolis UI

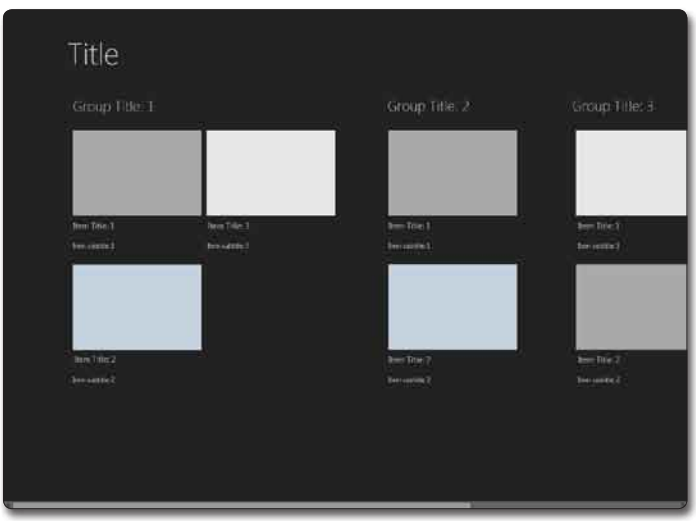


Figura 4. Formulário principal de uma aplicação Grid Metropolis UI

Figura 5. Formulário de detalhes de uma aplicação Grid Metropolis UI

Ainda em se tratando de termos visuais, o XE3 traz consigo alguns novos estilos, marcados como Metropolis UI, prontos para serem usados em aplicações deste tipo, seja em VCL ou FireMonkey, na forma de Skin. Mais precisamente são quatro as opções disponíveis: Metropolis UI Black, Metropolis UI Blue, Metropolis UI Dark e Metropolis UI Green (**Figura 6**). Apenas lembrando que a aplicação de determinado estilo não é perceptível em tempo de Design, só podendo ser visualizado em Runtime.

Convertendo uma aplicação existente para Metropolis UI

Conforme era de se esperar, o Delphi XE3, além de apresentar o novo recurso de Metropolis UI, ainda provê meios para a conversão de uma aplicação existente para este estilo, seja ela VCL ou FireMonkey. Tomando então como exemplo uma aplicação tradicional do tipo VCL Forms Application, no Form Designer, no menu de contexto do formulário é disponibilizada a nova opção "Convert to Metropolis UI Form". Uma vez acionada, ela exibe uma mensagem de alerta da conversão que está para se iniciar, informando que as alterações, caso sejam realizadas, não poderão ser revertidas. Ao confirmar em OK, o processo é iniciado, e o próprio Delphi se encarrega de fazer as mudanças necessárias, a fim de deixar o visual do formulário no estilo Metropolis UI. Dentre as alterações realizadas neste momento, está a retirada das bordas do Form (BorderStyle = bsNone) e o seu estado de janela como maximizado (WindowState = wsMaximized).

Por fim, as opções de projeto são automaticamente abertas para que seja selecionado o estilo Metropolis desejado (**Figura 6**). Assim que se define um Style para a aplicação, o Form está pronto para ser executado com a aparência Metropolis UI.

Visual LiveBindings

Surgindo como uma das grandes novidades da versão XE2, o LiveBindings é uma nova forma de se ligar dados no Delphi. Este tipo de recurso, aliás, já era habitual até então, através das ligações naturais estabelecidas entre DataSets e seus Fields, a controles do

tipo Data-Aware, os famosos "DB", tais como DBEdit, DBText, DBGrid, entre outros. Todavia, o LiveBindings vai além deste roteiro, com a intenção de permitir se ligar "qualquer coisa a qualquer coisa", popularmente falando. Disponível tanto para aplicações VCL quanto para FireMonkey, através de seu uso é possível, por exemplo, se conectar objetos dos mais variados tipos a controles em tela, um controle a outro, ou mesmo uma propriedade a outra propriedade do mesmo controle. Para que seu trabalho possa ser devidamente realizado, toda sua estrutura é baseada em expressões. Tais expressões são tratadas como Binding Expressions, e são utilizadas durante o processo de "Binding" entre os objetos ou as partes envolvidas. Essas "expressões de ligação" irão conter, por exemplo, as propriedades ou valores envolvidos no Binding.

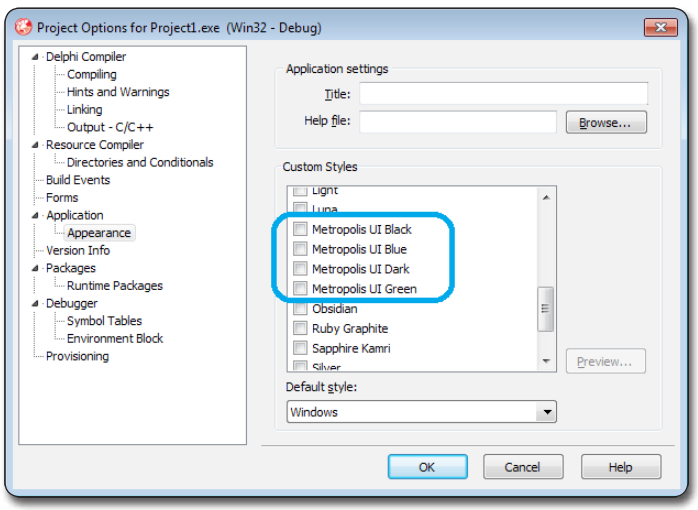


Figura 6. Novos estilos Metropolis

Dado o passo inicial, agora no Delphi XE3 este mecanismo foi apurado, de forma a trazer melhorias que visam principalmente colaborar na agilidade de seu uso. Prova disso, é que até então, para fazer uso efetivo do LiveBindings, havia essencialmente a necessidade do desenvolvedor escrever algum código. Seja para a definição das expressões utilizadas, ou pela notificação manual das mudanças de dados ocorridas ao mecanismo, o trabalho existia. Agora tudo isso ficou ainda mais fácil, através do Visual LiveBindings. Conforme seu próprio nome indica, este recurso provê uma forma totalmente visual de se criar e estabelecer as ligações mencionadas anteriormente. De acordo com este contexto, nesta versão do Delphi há duas novas formas de se definir LiveBindings, através do uso de duas novas estruturas: LiveBindings Designer e LiveBindings Wizard.

LiveBindings Designer

Agora no XE3, podemos estabelecer que a forma mais simples de se criar LiveBindings é através do uso do LiveBindings Designer, que pode ser definido com uma espécie de diagrama visual de Bindings. Isto porque, através da superfície de sua estrutura é possível criar, editar, excluir e visualizar os LiveBindings existentes em um formulário, de uma forma totalmente visual a intuitiva. No IDE, ele está disponível no menu View. Adicionalmente, ele pode ser acionado de forma automática durante a criação de um novo LiveBinding em formulários de projetos VCL e FMX, através da opção Bind Visually dos componentes. A **Figura 7** ilustra sua estrutura e seu modo visual de "ligar" dados de objetos.

No exemplo utilizado na imagem, foi estabelecida uma ligação entre os controles Edit1 e Label1, de forma que o que é digitado na caixa de texto é refletido no texto do Label, assim que o cursor sai do Edit. Para nos certificarmos da praticidade do recurso, a partir de uma aplicação VCL comum, basta arrastar para o formulário os controles mencionados e um Button, que será usado apenas para que o cursor saia da caixa de entrada ao se pressionar a tecla Tab. Acionando então o LiveBindings Designer, já em seu editor, através do simples movimento de arrastar-e-soltar do mouse da propriedade Text do Edit1 em direção à propriedade Caption do Label, define-se um novo LiveBinding. Ao executar a aplicação, podemos ver a funcionalidade descrita anteriormente. Mesmo se tratando de um exemplo totalmente trivial, podemos notar claramente as diferenças de usabilidade impostas pelo novo Visual LiveBindings. Na prática, por um simples movimento de mouse foi definido um novo LiveBinding, sem a necessidade de se codificar qualquer expressão ou notificação envolvida. Além disso, todas as operações e configurações realizadas no diagrama exposto no Designer são persistidas, ficando nativamente salvas na mesma pasta do projeto, em um arquivo de extensão .vbl.

Tanto o LiveBindings Designer quanto o LiveBindings Wizard utilizam o conceito de QuickBindings, que se reflete no estabelecimento de Bindings mais tradicionais. Em suma, eles se resumem a ligações envolvendo os controles em tela, tal como ligar um Grid a uma fonte de dados, uma propriedade de um controle a outro ou a um Field de um DataSet. Além disso, cada controle UI só pode ser definido uma vez como ponto de destino no contexto. Em outras palavras, tomando como base o exemplo anterior, caso fosse adicionado outro Edit ao Form, para se estabelecer um novo Binding com o Label existente, a ligação anterior com o Edit1 teria que ser excluída.

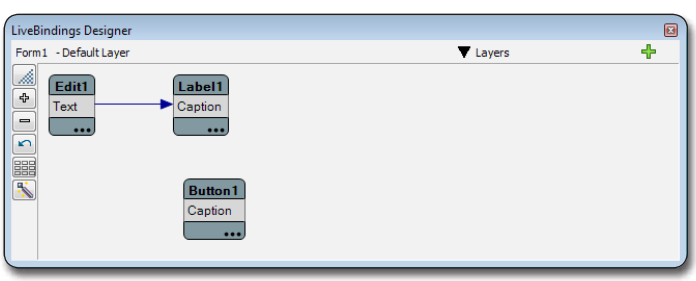


Figura 7. LiveBindings Designer

LiveBindings Wizard

Disponível também para projetos VCL e FireMonkey, o LiveBindings Wizard se apresenta na forma de um eficiente assistente para criação de Bindings. Para acessá-lo, você pode usar o próprio LiveBindings Designer, a partir de um botão de ação exclusivo.

Uma vez acionado, o Wizard é mostrado em tela, conforme mostra a **Figura 8**.

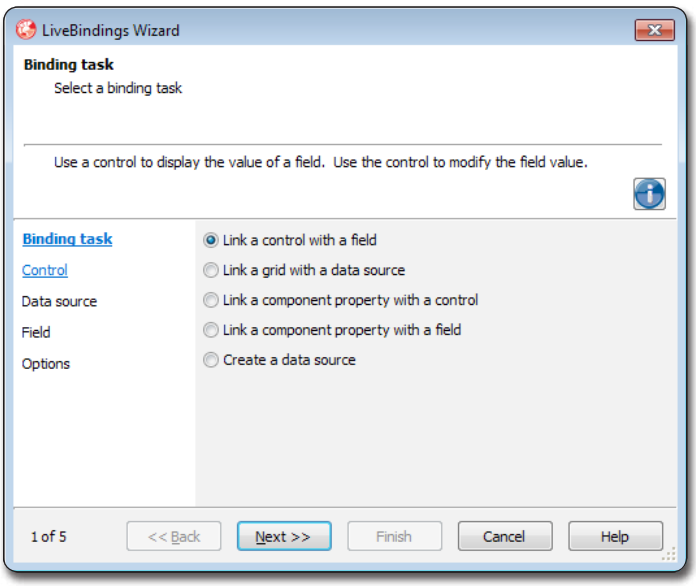


Figura 8. LiveBindings Wizard

Em termos de estrutura, a área central do LiveBindings Wizard pode ser dividida em duas partes principais. Do lado esquerdo, estão os passos envolvidos no assistente, enquanto que do lado direito estão as opções disponíveis naquele momento. Sendo assim, os passos envolvidos no assistente variam de acordo com a opção escolhida ainda no passo inicial. Portanto, esta primeira tela é peça-chave para a continuidade do processo. Sendo assim, analisando a página inicial mostrada na imagem, as seguintes opções relacionadas a Binding são providas:

- Vincular um controle a um campo (Field);
- Vincular um Grid a uma fonte de dados (Data Source);
- Vincular a propriedade de um componente a um controle;
- Vincular a propriedade de um componente a um campo (Field).

FireMonkey FM2

No Delphi XE2, sem sombra de dúvidas, a novidade que teve maior destaque em meio a comunidade e a grande mídia, antes mesmo de seu lançamento oficial, foi a nova plataforma de aplicações FireMonkey. Isto porque, naquele momento ela incorporava algo realmente novo e robusto ao Delphi, que é a possibilidade de criação de aplicações com um visual mais rico e flexível do que as tradicionais VCL Forms Application, e direcionadas tanto para plataformas Desktop (Windows e Mac OS X) quanto para móveis (iOS). O resultado disso era interfaces que proporcionavam uma melhor experiência do usuário, com suporte a efeitos, animações e multimídia (áudio e vídeo). Outro detalhe importante é que toda esta flexibilidade e riqueza de visual citados, se faz graças à sua outra característica singular, que é o de fazer proveito da unidade de processamento gráfico do Hardware para otimização e independência de sua parte gráfica.

Sendo assim, seguindo uma evolução natural, a nova versão do FireMonkey no XE3 é então apresentada como FireMonkey FM2. Dentre suas novidades, a mais perceptível diz respeito à mudança de foco da tecnologia. Isto porque, a plataforma agora é direcionada exclusivamente ao desenvolvimento Desktop, não provendo mais suporte à dispositivos móveis, ou seja, ao iOS. A justificativa para isso é a elaboração, por parte da Embarcadero, de um novo produto exclusivo ao desenvolvimento Mobile, intuitivamente nomeado como RAD Studio Mobile e com previsão de lançamento para 2013 (ver Roadmap do produto na seção de **Links**).

Além disso, assim como a VCL, o FireMonkey também traz como novidade o seu suporte ao desenvolvimento de aplicativos que seguem o estilo e a experiência de usuário do Windows 8, e que irão rodar tanto em Desktops quanto em Tablets. Sua parte Mac também ganha acréscimos, visando atender de forma nativa o também recente OS X Mountain Lion. Visando atender estas duas plataformas (Windows e Mac) de maneira satisfatória, o FireMonkey provê agora suporte a Touch e Gestures, ou seja, toques e gestos, assim como já acontecia na VCL. Talvez a única grande diferença seja o fato dos Gestures do FireMonkey ainda não suportarem a customização de gestos (Custom Gestures), sendo necessário utilizar somente os nativos. A **Figura 9** mostra como exemplo a propriedade relacionada a Gestures de um simples formulário FMX.

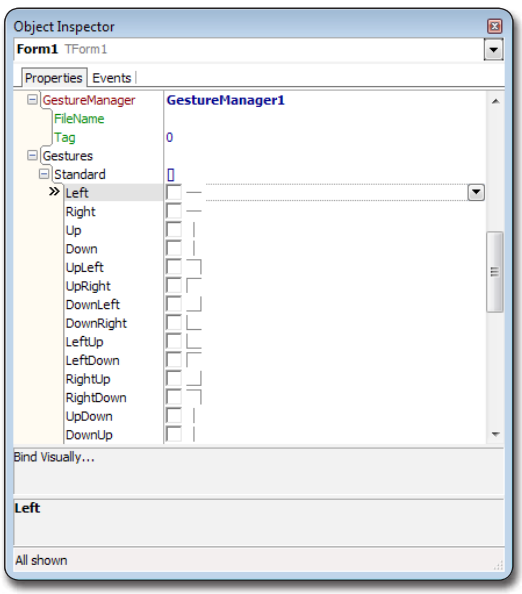


Figura 9. Propriedades relacionadas a Gestures de um formulário FireMonkey

Bitmap Style Designer

Ainda na versão XE2, foi incluso no Delphi o conceito de Styles, ou seja, estilos, que se traduzia no recurso de VCL Styles. Um estilo, quando aplicado a uma aplicação VCL Forms era capaz de mudar sua aparência, definindo um novo conjunto de detalhes gráficos, na mesma medida que um tema do Windows. Apesar de ter surgido como uma das novidades do XE2, tal recurso já era muito conhecido e utilizado em meio à comunidade, através dos

famosos componentes de Skins, produzidos por outras empresas. Adicionalmente ao até então novo recurso de VCL Styles, foi apresentado também um eficiente Designer de estilos, denominado VCL Styles Designer, onde o desenvolvedor poderia estar criando e testando seus próprios Styles.

Seguindo este contexto, agora associado tanto à parte VCL quanto FMX está o novo Bitmap Style Designer. Esta estrutura nada mais é do que o anterior VCL Style Designer do XE2 com um nome totalmente diferente em razão da expansão de seus recursos e usabilidade. Agora ele pode ser usado tanto para estilos VCL como também para estilos FireMonkey. Sendo assim, sua finalidade principal é a de atender toda a parte de criação, edição e testes de estilos (Styles) para aplicações Delphi. Disponível no menu Tool do IDE, o Bitmap Style Designer pode ser visto na **Figura 10**.

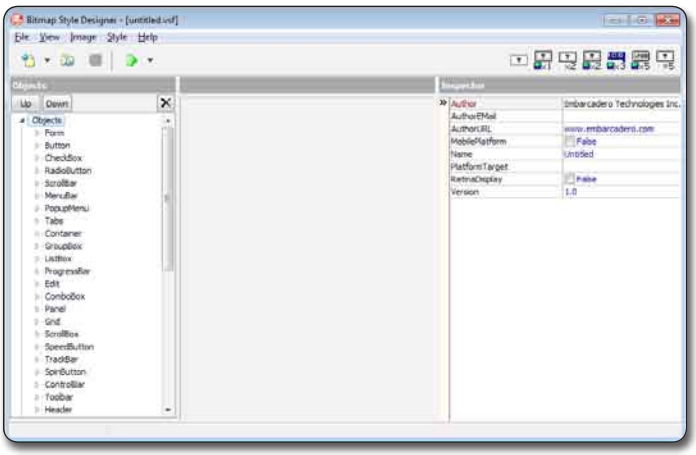


Figura 10. Bitmap Style Designer

Entitlement List e Provisioning para aplicações Mac OS X

Entitlement List e Provisioning são os nomes de duas das novidades exclusivas ao desenvolvimento Mac OS X com FireMonkey. A primeira, Entitlement List, diz respeito à lista de "direitos" que a aplicação deverá ter, tal como se ela terá ou não permissão de leitura e escrita a determinadas pastas do sistema (Ex: Movies, Music e Pictures), direito a impressão ou interação com dispositivos USB, etc. A lista completa com todos os possíveis "direitos" fica disponível no menu Project>Options>Entitlement List, conforme mostra a **Figura 11**.

Já o termo de Provisioning diz respeito ao processo de provisionamento de uma aplicação Mac OS X. No IDE, o Provisioning nada mais é que uma opção de projeto que provê alguns campos de dados onde deverão ser informados, por exemplo, o seu perfil de desenvolvedor (Mac Developer Profile). Esta informação fica então relacionada a um tipo de Build denominado OSXSandBox (ver **Nota do DevMan 2**), necessário para que sua aplicação possa ser enviada à Mac App Store. Assim como Entitlement List, o Provisioning fica disponível no menu Project>Options (**Figura 12**).

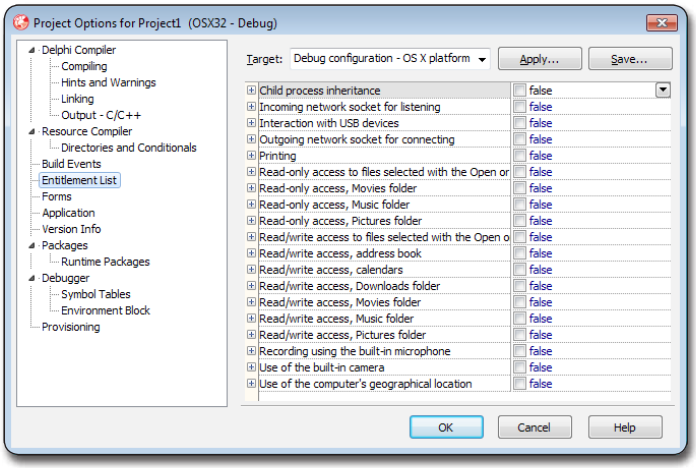


Figura 11. Entitlement List

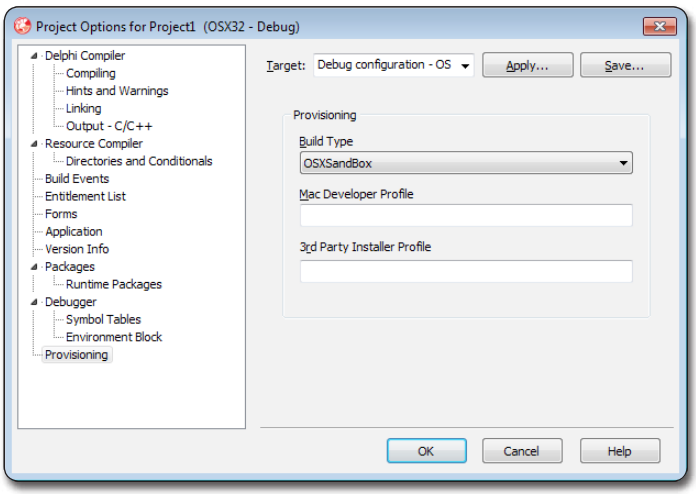


Figura 12. Provisioning

Nota do DevMan 2

OSXSandBox é um tipo de Build relacionado ao recurso de SandBoxing (algo como caixa de areia, numa tradução livre para o português) da Apple, que cria um ambiente único e isolado para cada aplicativo Mac, de forma a certificar que eles realizem operações somente dentro de suas designações. Em outras palavras, neste ambiente, um aplicativo não consegue acessar e manipular dados de outros aplicativos, estabelecendo assim um robusto critério de segurança. Anteriormente ao SandBoxing, os limites para um aplicativo Mac se resumiam apenas em permissões de usuário, o que dava brechas para a atuação de Softwares mal-intencionados (Malwares). Sendo assim, um simples aplicativo Mac poderia, entre outras cosias, ter acesso ao conteúdo de outros aplicativos, bem como coletar dados do próprio usuário.

Version Info contendo três Build Options

Version Info é um recurso que já acompanha o Delphi de longa data, permitindo que se especifiquem diversas informações relacionadas à versão de projetos VCL e FireMonkey. Tais informações ficam então disponíveis nas propriedades do aplicativo gerado, acessadas pelo menu de contexto Propriedades, através do botão direito do mouse. Além disso, é possível ainda se incluir o número da versão da aplicação em meio às suas informações. O incremento

desta numeração pode se dar de forma automática a cada Build do projeto. No XE3 agora são três as opções de incremento:

• Do not change build number: o número do Build não é alterado;

• Auto generate build number: gera e incrementa a numeração de Release e Build automaticamente;

• Auto increment build number: incrementa a numeração de Build a cada processo de Build do projeto;

Outras novidades

Conforme já dito, várias outras melhorias e novidades de menor proporção também foram apresentadas com o Delphi XE3. Dentre estas, algumas valem destaque, tais como:

• Novos drivers dbExpress para SQLite e InterBase ToGo. Tais bancos de dados se assemelham pelo fato de serem voltados para o desenvolvimento de aplicações de pequeno porte, uma vez que não exigem a presença de um servidor de banco de dados. Isto porque, eles atuam tanto no papel de cliente quanto de servidor, rodando em um processo único de execução. Por este fato, ambos podem ser facilmente "embutidos" diretamente nas aplicações;

• Adição de novos Wizards de projetos, tal como para as novas e já mencionadas aplicações Metropolis UI (VCL e FireMonkey), bem como também para aplicações tradicionais FireMonkey, agora tratadas como FireMonkey Desktop Application. Sendo assim, no XE3, ao iniciar uma nova aplicação FMX, um assistente é aberto para que seja selecionado o tipo de aplicação desejado, dentro do contexto FireMonkey – HD Application ou 3D Application **(Figura 13)**;

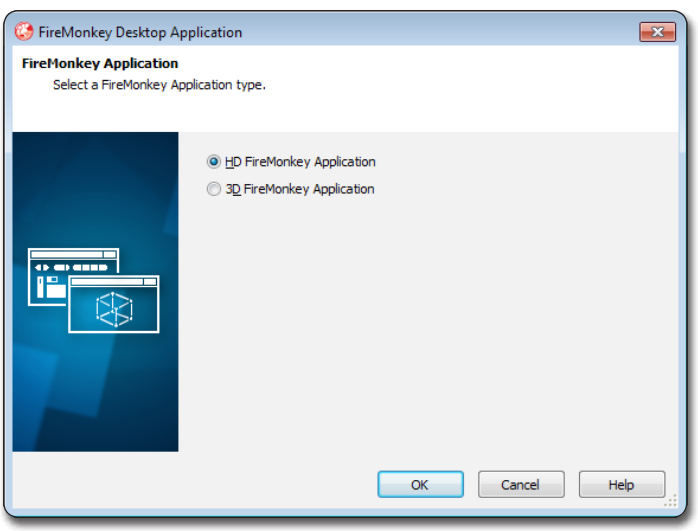


Figura 13. Selecionando o tipo de aplicação FireMonkey Desktop Application

Além disso, a família XE3 conta ainda com outros adicionais externos, mais especificamente outros produtos, que de certa forma ainda estão relacionados ao Delphi. Tais produtos inclusive já são bem conhecidos da comunidade, conforme será visto a seguir.

HTML5 Builder

Como sabemos, o RAD Studio é uma solução completa de desenvolvimento e que envolve vários produtos da Embarcadero. Dentre estes, o Delphi se apresenta como destaque, acompanhado de outras ferramentas, tais como o C++ Builder, Prism (.NET) e o Rad PHP. Estes dois últimos, inclusive, já foram tema de vários artigos aqui mesmo nesta revista. Em meio às novidades apresentadas na versão XE3, o Rad PHP muda de nome, sendo apresentado agora como HTML5 Builder (**Figura 14**), numa clara referência ao novo foco do produto, o badalado HTML5. Além disso, dada à grande tendência de aplicações para Web e Mobile, o produto, ainda em sua versão passada (XE2) já dava mostras em atender este segmento, provendo recursos suficientes para a construção de aplicativos para internet e dispositivos móveis (Ex: BlackBerry e Android). Agora no HTML5 Builder esta característica se renova pela adoção de um padrão de código totalmente baseado em HTML5, CSS3 e JavaScript, além do próprio PHP. A intenção é que você, a partir de uma única base de código, crie seu aplicativo uma única vez e faça seu Deploy para múltiplas plataformas, seja ela Web ou Mobile.

Apoiado em um Framework fortemente visual, no HTML5 Builder é disponibilizada uma série de componentes visuais, tal como no Delphi. Sendo assim, pelo simples arrastar-e-soltar de componentes em tela, é possível construir e personalizar suas aplicações de forma rápida e eficiente. Prova disso, é o novo componente visual de geolocalização, que encapsula uma API específica do próprio HTML5, tornando a vida do desenvolvedor mais simples. Além disso, isto faz com que você não tenha que necessariamente aprender uma nova linguagem ou tecnologia, tal como JavaScript e suas bibliotecas (Ex: jQuery e jQueryMobile). Adicionalmente, ainda graças ao HTML5, a ferramenta provê recursos mais poderosos de multimídia, através de componentes para áudio e vídeo a serem utilizados em aplicações Web, e que excluem a dependência de plug-ins extras, tal como o Adobe Flash.

Prism XE3

Outro produto muito próximo do Delphi, conforme já dito, é o Prism, surgido inicialmente como Delphi Prism. Dentre suas características, estão o seu formato de Plug-in para a principal IDE .NET, o Microsoft Visual Studio, além do uso de uma linguagem muito próxima do Delphi, também baseada em Pascal, denominada Oxygene. Seu outro ponto atrativo está relacionado à sua flexibilidade em prover o desenvolvimento de aplicações para plataformas que vão além do Windows, tais como Linux e Mac OS X. Isto porque, o Prism suporta, além do Microsoft .NET, outro Framework denominado Mono, que atua como uma versão "multi-plataforma" do mesmo .NET.

Na família XE3, o Prism também ganha sua nova versão, apresentando novidades principalmente relacionadas ao desenvolvimento de aplicações nativas para WinRT, a mais nova API do recente Windows 8. Ainda seguindo pelas novidades do

Prism, o Shell utilizado em sua instalação é agora baseado no Visual Studio 2012, o que faz com que o produto possa desfrutar dos recursos mais atuais do Microsoft .NET Framework. A **Figura 15** mostra um esboço deste novo ambiente do Prism XE3.

Conclusão

Vimos neste artigo, de forma geral, as principais novidades apresentadas no Delphi XE3. Como acontece em toda nova versão, pudemos notar a melhoria de recursos existentes, tal como a nova forma de criar Bindings através do novo Visual LiveBindings, bem como o acréscimo de novos recursos, como as aplicações Metropolis UI. Ao mesmo tempo tomamos ciência da preocupação da Embarcadero em manter o produto em sintonia, mesmo que de forma sucinta, com o que há de mais novo no mercado. Prova disso é o provimento de recursos que habilitem a construção de aplicações Delphi para as novas versões dos Sistemas Operacionais suportados, vide Windows 8 e Mac OS X Mountain Lion.

Novamente, é importante frisar que o objetivo deste artigo foi o de expor o maior número de informações possíveis sobre o que há de mais novo nesta mais recente versão do Delphi. Além disso, serve também como o "Start" para os futuros artigos relacionados a cada uma das novas funcionalidades deste Release, tratadas então de forma mais específica.

Agradeço a todos e até a próxima.

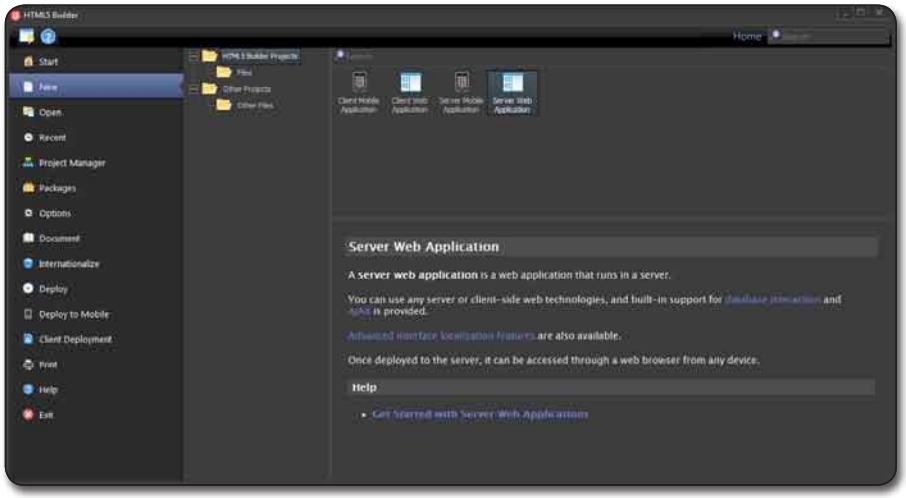


Figura 14. HTML5 Builder

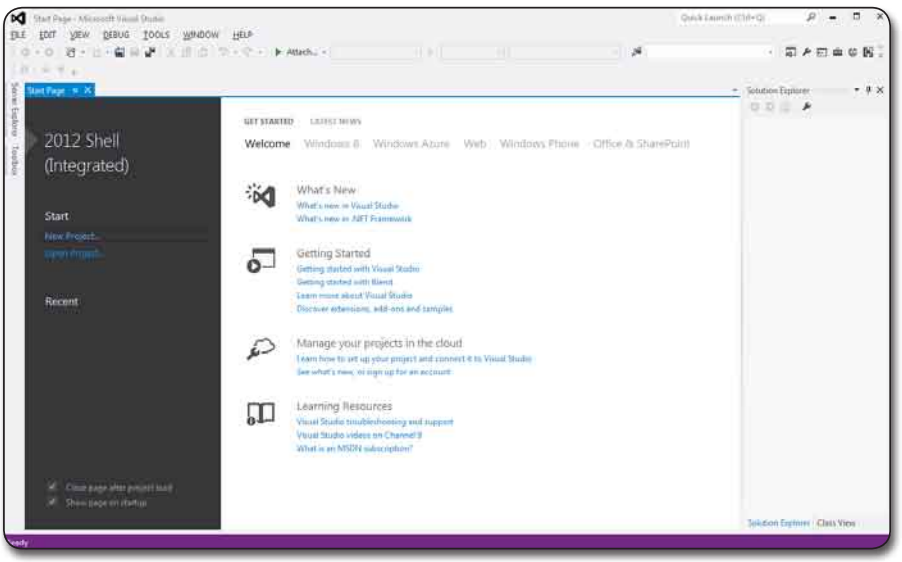


Figura 15. Prism XE3

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 7 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Blog: prismsilver.blogspot.com

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Delphi XE3 – Página oficial do produto
http://www.embarcadero.com/products/delphi

Delphi XE3 – Free Trial
http://www.embarcadero.com/products/delphi/downloads

Mono - Página oficial do projeto
http://www.mono-project.com/Main_Page

RAD Studio Mobile – Roadmap
http://edn.embarcadero.com/article/42544

HTML5 Builder - Página oficial do produto
http://www.embarcadero.com/products/HTML5-Builder

RIA com Delphi – Parte 1

Aprenda criar aplicações ricas sem dominar HTML

Este artigo faz parte de um curso

A Web é um ambiente que cada vez mais se faz presente no cotidiano das pessoas. Atualmente, é difícil imaginar um mundo sem a Internet, afinal ela se tornou a base para muitas profissões e o mecanismo chave para possibilitar a transferência de informações de uma forma praticamente instantânea entre as organizações. Diante deste cenário, deixar de desenvolver aplicações para a Web significa não explorar uma plataforma potencialmente lucrativa e extremamente importante. Entretanto, vale à pena lembrar que uma aplicação dentro deste contexto não se destina apenas a Internet, mas pode ser utilizada da mesma forma em uma rede interna da empresa, assim como os tradicionais Softwares Desktop são, trazendo muitos benefícios de uma forma geral. Além do mais, o protocolo padrão HTTP (HyperText Transfer Protocol), um dos principais da Internet, é uma das bases que possibilitam a criação de aplicações heterogêneas, rompendo a barreira de incompatibilidades imposta pelos diversos Sistemas Operacionais, onde uma aplicação cliente que é baseada neste padrão, operando através de um navegador, pode rodar satisfatoriamente no ambiente Windows, Linux, Android, e assim por diante. Uma aplicação Web normalmente é baseada em uma arquitetura multicamadas, composta por tecnologias server-side e client-side, trazendo muitas vantagens, como é o caso da otimização de futuras atualizações e manutenções do Software. No server-side, além do Sistema Operacional, deve existir o servidor HTTP, cuja finalidade é oferecer a base para a utilização e gerenciamento de todos os recursos envolvidos que são acessados e consumidos, possibilitando as requisições e respostas, onde tal servidor pode ser um

Resumo DevMan

RIA com Delphi:

A Internet já é parte da vida de muitas pessoas. Está cada vez mais presente nos dispositivos móveis e é claro, nos computadores das empresas. Evidentemente este ambiente é muito promissor e merece ser acompanhado de perto. Quando o assunto é o desenvolvimento de aplicações para a Web, de início pode haver certa preocupação, pois, diversas tecnologias e fatores são necessários para que este objetivo seja atingido. Contudo, vale à pena enfatizar que não são aplicações apenas para Internet, mas que também podem ser utilizadas pelos usuários na rede interna da Empresa, através de navegadores. Visando minimizar os esforços e estender as possibilidades quando o assunto é RIA, surgiram diversos Frameworks e ferramentas que auxiliaram este processo. O uniGUI e o Raudus são exemplos disto. Sendo componentes para o Delphi, possibilitam o desenvolvimento RIA de uma forma extremamente simples e rápida, onde nem existe a obrigatoriedade da equipe envolvida dominar HTML, uma das principais linguagens quando o assunto é o desenvolvimento para a Web. Este primeiro artigo aborda o conceito básico para o contexto RIA, apresentado as características e os principais recursos envolvidos em uma aplicação Web, demonstrando posteriormente, o componente uniGUI, um excelente Framework para elaborar estas aplicações com o Delphi, através de um simples exemplo que emprega uma aplicação autônoma.

Em que situação o tema é útil:

Este tipo de Framework possibilita a criação de aplicações para a Internet ou uma rede interna (Intranet) mesmo sem domínio dos recursos envolvidos, como é o caso de linguagens amplamente utilizadas para este tipo de desenvolvimento como é o caso do PHP, HTML e Javascript, utilizando excelentes Frameworks e componentes da atualidade que simplificam todo o trabalho, aproveitando os diversos recursos do Delphi XE 2 e podendo empregar todo o conhecimento já adquirido através do RAD.

IIS (Internet Information Services), um Apache, etc. Além do servidor HTTP, também deve existir um recurso para processar e tratar tais solicitações e retornos, uma linguagem de programação server-side, como é o caso do PHP, ASP, Java (JSP), dentre outras, normalmente interligadas a um ou mais sistemas de gerenciamento de banco de dados (SGBDs). O servidor também pode contar com políticas de segurança, como é o caso de um Firewall, um recurso implementado através de Software ou Hardware para bloquear acessos indevidos a áreas especificas, liberando apenas as funcionalidades que podem ser utilizadas publicamente, podendo ainda efetuar controles mais rigorosos a níveis de usuário e máquina, por exemplo. Do lado cliente, basta a existência de um navegador (Browser) para a exibição da página HTML (HyperText Markup Language) que é gerada pelos recursos do servidor de aplicação, em outras palavras, a interface gráfica da aplicação Web para a interação com os usuários. Aí está um grande diferencial, apenas a necessidade de um Browser, facilitando muito o trabalho, pois, não há a necessidade de provedores de acesso ao banco, instalações de Softwares adicionais, dentre outros recursos que podem ser requeridos em aplicações Desktop tradicionais e geram tarefas extras. Para uma maior interatividade e desempenho no client-side, a linguagem JavaScript também pode ser empregada, tratando as informações, objetos, eventos, requisições e respostas de uma forma mais eficiente, utilizando recursos como AJAX (Asynchronous Javascript and XML), JSON (Javascript Object Notation – **ver nota do DevMan 1**), dentre outros, permitindo, por exemplo, funcionalidades importantes como requisições assíncronas ao servidor, pré-validações e mascaramento de campos. Além do mais, existe uma diversidade de Frameworks e plug-ins para Javascript que aumentam consideravelmente a produtividade e interatividade, como é o caso da biblioteca jQuery, muito empregada em aplicações dentro deste contexto e sites. Outro recurso importante no client-side é a linguagem CSS (Cascading Style Sheets), cujo principal intuito é permitir a definição de estilos para a formatação de elementos presentes nas páginas HTML, mas que também possibilita o emprego de outros recursos, como é o caso de animações. A **Figura 1** apresenta uma simples ilustração de uma estrutura que pode ser utilizada em uma aplicação Web.

Nota do DevMan 1

A notação JSON (JavaScript Object Notation) é o padrão adotado para a definição de objetos na linguagem client-side JavaScript, porém não está limitada apenas a esta linguagem, podendo a mesma ser empregada em diversas ferramentas de desenvolvimento, como é o caso do próprio Delphi, através da Unit DBXJSON. Tem sido muito utilizada atualmente para a transferência de informações e armazenamento de dados simples, pois, é mais leve e rápida que o XML, uma vez que é mais enxuta. Sendo assim, o JSON também pode ser empregado em conjunto com AJAX para obter um melhor desempenho nas requisições assíncronas.

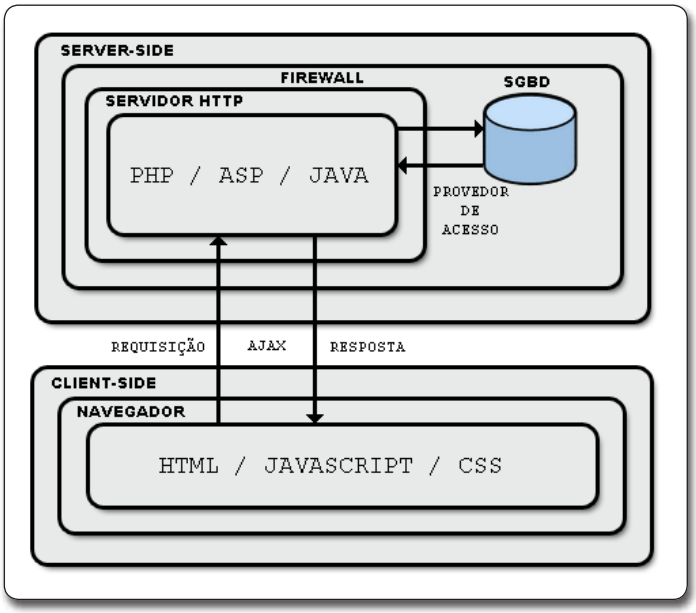


Figura 1. Exemplo de estrutura de uma aplicação Web

Sendo assim, uma aplicação Web é resultante da utilização de diversas tecnologias empregadas em conjunto, podendo se tornar uma tarefa desafiadora para a equipe de desenvolvimento, além de consumir um tempo considerável.

Em meio às aplicações Web, surgiu o termo RIA (Rich Internet Application – Aplicações Ricas para a Internet). Apresentado primeiramente pela Macromedia, hoje Adobe, RIA é utilizado para descrever uma aplicação para a Internet que proporciona uma experiência ainda mais rica para os usuários, aumentando assim sua satisfação, interação e produtividade, possuindo características semelhantes aos Softwares da plataforma Desktop, podendo ainda ser definida como uma evolução dos tradicionais sistemas para a Web. Uma aplicação RIA pode ser desenvolvida nas mais diversas tecnologias, Frameworks e linguagens de programação, sendo também mantida em um servidor de aplicação e projetada para ser compatível com todos ou grande parte dos navegadores (cross-browser) e empregar comunicação assíncrona com o servidor, que pode ser realizada através de AJAX ou outros recursos de acordo com a ferramenta adotada para o projeto.

Devido a suas características, tal como o fato da aplicação RIA executar em um servidor (server-side), a mesma também não necessita de instalação nas máquinas clientes, uma vez que os navegadores são utilizados, salvo que pode haver a necessidade de algum plug-in adicional para o Browser em poucos casos, que normalmente é fornecido pela própria aplicação.

Como pode ser observado, o desenvolvimento de uma aplicação Web pode ser uma tarefa complexa pelo fato de englobar diversas tecnologias, onde a manipulação de sessões e cookies, o gerenciamento de conteúdo, as atualizações dinâmicas e toda a interatividade com o usuário devem ser minuciosamente trabalhados.

Diante deste fator, existem diversos frameworks que auxiliam e simplificam esta tarefa, minimizando os esforços e aumentando a produtividade.

Um dos ambientes de desenvolvimento para RIA de grande popularidade e extensamente empregado na atualidade é o Flash/Flex da Adobe. A plataforma Flash oferece suporte a algo muito além do que os tradicionais banners, jogos e animações presentes na Web, e pode ser utilizada facilmente para o desenvolvimento de aplicações ricas para a Internet de uma forma muito rápida e prática, através dos recursos e componentes que fazem parte da suíte de ferramentas.

Entretanto, também existe outro Framework importante muito adotado, o ExtJS. Baseado na linguagem JavaScript, o ExtJS é compatível com uma diversidade de navegadores e foi criado inicialmente com o intuito de ser uma extensão do Yahoo. É um Framework gratuito sob a licença LGPL extensamente documentado, mas também possui versões pagas que contam com algumas regalias, como é o caso de suporte, correções e atualizações mais recentes. O ExtJS utiliza recursos baseados em JSON e XML (Extensible Markup Language – **Nota do DevMan 2**) para tratar respostas, possibilitando o emprego de AJAX, dentre outros recursos, e permite o desenvolvimento de aplicações eficientes com interfaces gráficas de alta qualidade com grande performance, podendo ainda ser integrado a outros Frameworks, como é o caso da biblioteca jQuery, mencionada anteriormente.

Atualmente, o Framework ExtJS possui diversas versões gratuitas (ver seção links), empregadas diretamente através da linguagem HTML, e extensões, que podem ser adicionadas a várias ferramentas de desenvolvimento.

Nota do DevMan 2

O XML (Extensible Markup Language) é uma linguagem de marcação padrão baseada em texto, assim como a HTML, sendo também constituída por Tags, porém não existe uma pré-definição para as mesmas, qualquer palavra pode ser empregada, salvo algumas exceções. Os documentos XML são muito utilizados no ambiente da Internet para transferência de informações, bem como o armazenamento de determinados dados, como é o caso da NF-e. Apesar de não possuir uma limitação para Tags, deve ser mantido um padrão para a estrutura do arquivo, que por sua vez, é composta por nós que derivam de um elemento raiz e que podem ter quantos nós filhos forem necessários para o documento em questão.

Com relação ao Delphi XE 2 e algumas versões anteriores, existe um Framework que permite o desenvolvimento de RIA e Desktop em uma única aplicação, o uniGUI (unified Graphical User Interface), criado pela FMSoft. Ele possui um conjunto de componentes que podem ser adicionados facilmente ao RAD, tais como painéis, botões, labels, edits, etc. Sendo assim, é possível projetar, elaborar todo o código fonte, testar e depurar a aplicação empregando todos os recursos do Delphi e, posteriormente, utilizar tal aplicação para a Web sem que o desenvolvedor tenha conhecimento e domínio de outras linguagens que normalmente seriam necessárias na elaboração de uma aplicação RIA, tais como HTML e Javascript. O uniGUI é baseado na biblioteca apresentada anteriormente, ExtJS, portanto um servidor uniGUI requer esta biblioteca, que por sua vez, acompanha o conjunto de ferramentas. Além do Software Desktop e RIA em uma mesma aplicação, o uniGUI também permite a elaboração de módulos ISAPI, que podem ser utilizados em servidores Apache ou IIS (**Notas do DevMan 3 e 4**) e será melhor detalhado posteriormente, bem como a criação de um serviço de aplicação para o Windows.

O Delphi também possui outros Frameworks e componentes que podem ser utilizados para o desenvolvimento de RIA, e outro que merece destaque é o Raudus. Disponível também para o Lazarus, o Raudus é um exemplo de ferramenta que é gratuita e permite a elaboração de um Software Desktop e RIA em uma única aplicação. Ele possui sua própria biblioteca front-end. Através de seu WebKit offscreen, ele cria automaticamente a interface gráfica com visual baseado em WYSIWYG (What You See Is What You Get – O que você vê é o que você obtém), de acordo com a aparência do Software Desktop, para a execução da aplicação através de vários navegadores, em outras palavras, a aplicação RIA, tendo um alto desempenho. O Raudus também emprega os recursos de AJAX para as requisições assíncronas, onde a página principal não é recarregada. Também possibilita a criação de componentes próprios através de HTML5, Javascript e CSS3, embora não seja uma obrigatoriedade. Além da aplicação autônoma Desktop e RIA, o Raudus também possibilita a criação de módulos para um servidor Apache ou uma aplicação FastCGI, outras funcionalidades interessantes que ampliam as possibilidades. Entretanto, o Raudus será melhor apresentado posteriormente.

Como é possível perceber, as aplicações RIA são de extrema relevância nos dias de hoje e o desenvolvimento das mesmas pode ser uma tarefa simplificada através dos Frameworks que podem ser utilizados em conjunto com o Delphi, onde o desenvolvedor não necessita aprender novas linguagens de programação para desenvolver ricas aplicações para a Internet e consequentemente, outras plataformas.

Nota do DevMan 3

O IIS (Internet Information Services) é um Software servidor Web, desenvolvido pela Microsoft, que oferece suporte a linguagem ASP (Active Server Pages), mas também permite o emprego de outras tecnologias, gerando páginas HTML dinâmicas. Suas funcionalidades podem ser estendidas através do emprego de módulos ISAPI (Internet Server Application Programming Interface), que são bibliotecas adicionais que permitem o emprego de filtros ou ainda podem conter toda a aplicação Web ou RIA.

Nota do DevMan 4

O Apache também é um servidor Web (HTTP), porém é Open-Source, o que o tornou um Software muito popular na atualidade, possuindo versões para diversos Sistemas Operacionais. Assim como o IIS, ele também possibilita o carregamento de módulos adicionais que permitem estender suas funcionalidades, oferecendo suporte as mais variadas linguagens de programação.

Tutorial

uniGUI e sua instalação

Como mencionado, o uniGUI é um Framework composto por diversos componentes visuais e não-visuais que podem ser empregados no desenvolvimento rápido de aplicações RIA através do Delphi. Quanto à estrutura dos Softwares criados neste Framework, existem basicamente três tipos, sendo eles Standalone (autônomo), módulo ISAPI e aplicação de serviço do Windows. No modo autônomo (Standalone), o uniGUI possibilita que sejam criadas aplicações baseadas na biblioteca VCL que são um servidor Web ao mesmo tempo, o que implica que a aplicação RIA só pode ser acessada enquanto o Software está em execução. Tal servidor embutido na aplicação VCL permite que sejam criadas várias sessões, em outras palavras, diversos usuários podem acessar a aplicação RIA simultaneamente através dos navegadores. No modo módulo ISAPI, a aplicação baseada no uniGUI pode ser executada em um servidor IIS ou Apache. Um módulo ISAPI, Internet Server Application Programming Interface, uma extensão de servidor de Internet, é compilado como uma DLL, sendo carregado, controlado e chamado pelo servidor HTTP (IIS ou Apache). Já no modo de serviço de aplicação para o Windows, o Software é muito semelhante ao Standalone VCL, porém não é apresentada uma interface gráfica para a aplicação Desktop, ela é "apenas" um serviço e pode ser visualizada através do gerenciador de tarefas do Windows, mas também é exibido um ícone junto à barra de tarefas indicando que a mesma está em execução, permitindo também a abertura de um painel de controle para o aplicativo através de um clique duplo.

O uniGUI pode ser facilmente obtido diretamente do site do projeto em forma de executável (ver seção **Links**). Sua instalação segue o padrão tradicional (Next / Next / Finish), onde o desenvolvedor pode optar pelo diretório alvo e selecionar as versões do Delphi na qual deseja instalar o Framework. Contudo, uma vez instalado, o uniGUI precisa ser compilado no Delphi. Sendo assim, com o RAD aberto, basta acessar o diretório onde o mesmo foi instalado, cujo padrão é "C:\Program Files (x86)\FMSoft\Framework\uniGUI" para o Windows 7, e abrir o grupo de projeto referente a versão do Delphi em questão, como por exemplo, "uniGUI2012Group.groupproj". Todos os pacotes devem ser compilados, começando pelos que iniciam com "SynEdit" e seguindo a sequência em que os mesmos são apresentados na aba "Project Manager". Após a compilação de todos, basta instalar os pacotes para tempo de Design de acordo com a ordem apresentada a seguir:

SynEdit_D20...
uniGUI...dcl
uniGUI...Exdcl
uniGUI...Chartdcl

Concluída esta etapa, o conjunto de ferramentas do uniGUI está integrado ao RAD e já pode ser utilizado facilmente na elaboração das aplicações RIA.

uniGUI na prática

Para dar início a criação de uma aplicação RIA através do Delphi, será utilizado o assistente do uniGUI. Sendo assim, o menu File>New>Other deve ser acessado para abrir a janela "New Items". É possível notar que foi criado um novo grupo denominado "uniGUI for Delphi". A **Figura 2** apresenta os itens adicionados após a instalação do Framework uniGUI.

A próxima etapa é selecionar o item "Application Wizard" deste grupo e dar sequência pressionando o botão OK. Será aberta a janela do assistente do uniGUI para que o desenvolvedor defina as funcionalidades que deverão ser empregadas no projeto. A **Figura 3** apresenta a janela do assistente, "uniGUI Wizard Application".

Sendo assim, a opção selecionada para este exemplo foi "VCL Application / Standalone Server", que possibilita a criação da aplicação RIA e do Software Desktop baseado na biblioteca VCL

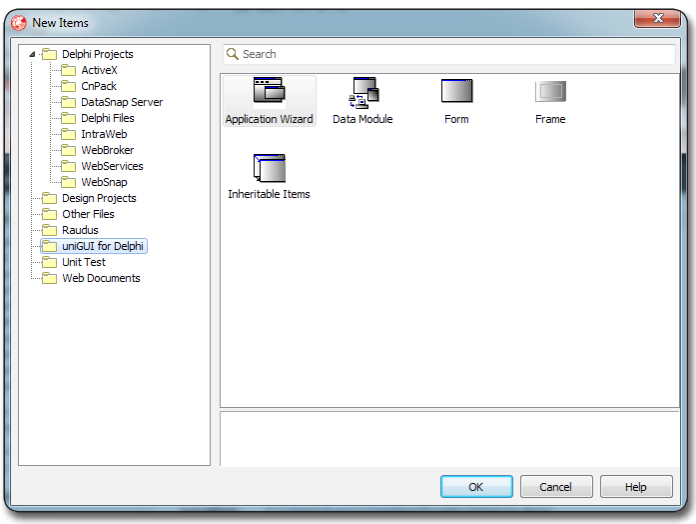


Figura 2. Janela "New Items" com o grupo "uniGUI for Delphi" em evidência

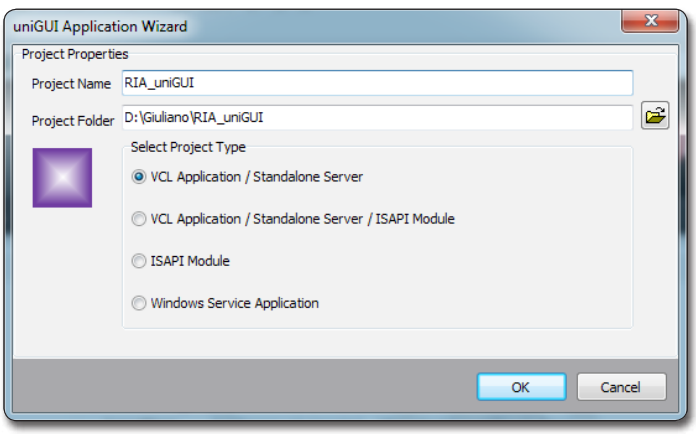


Figura 3. Janela do assistente do uniGUI

em um mesmo executável, uma aplicação autônoma. Também foram definidos o diretório e o nome para o projeto, "RIA_uniGUI". Após clicar no botão OK, basta confirmar a pergunta seguinte para habilitar o Framework VCL para o projeto. Posteriormente, serão criados automaticamente os três objetos básicos para a aplicação, o MainForm (TUniForm), o UniMainModule (TUniGUIMainModule) e o UniServerModule (TUniGUIServerModule).

Um ponto extremamente importante na elaboração da aplicação baseada no uniGUI é definir o diretório em que o Framework ExtJS se encontra. Como este Framework deve estar disponível para que a aplicação RIA funcione, deve ser especificado um diretório válido para o mesmo. A propriedade responsável por efetuar esta definição é denominada "ExtRoot", presente no objeto UniServerModule. Por padrão, seu valor é "[ext]\", que por sua vez aponta para o diretório "<Local de Instalação>\FMSoft\Framework\uniGUI\ext-...". Sendo assim, é conveniente alterá-la caso a máquina que hospedará a aplicação for diferente da máquina em que a mesma foi desenvolvida, o que ocorre na maioria dos casos. Diante deste fator, basta copiar o diretório ext da pasta de instalação do uniGUI para o mesmo diretório onde se encontra o executável da aplicação, por exemplo, e definir a propriedade "ExtRoot" em branco para que o uniGUI busque automaticamente o diretório ext na pasta onde o executável se encontra, e assim então distribuir ambos em conjunto, embora possa ser mais interessante, por questões de segurança, colocar tal pasta em um diretório com permissões somente de leitura no servidor. Para este exemplo, a propriedade foi mantida com seu valor padrão, "[ext]\".

Outro ponto importante para uma aplicação uniGUI é a definição das propriedades ServerRoot e CacheFolder, ambas do objeto UniServerModule. A ServerRoot é utilizada para atribuir o caminho raiz para todos os diretórios relativos. Já a CacheFolder, define a pasta temporária para a criação automática dos arquivos necessários. Vale à pena ressaltar que, por padrão, ambas estão definidas em branco, o que implica que o uniGUI utilizará o mesmo diretório em que se encontra o executável. Sendo assim, esta propriedade também não foi alterada para o exemplo, mas pode ser levada em consideração quando a plataforma alvo da aplicação for outra máquina.

Para que a aplicação RIA seja acessada através de navegadores, também deve ser especificada uma porta TCP (Transmission Control Protocol – **ver nota do DevMan 5**) para a mesma, lembrando que portas são utilizadas para que diversos Softwares distintos possam estabelecer comunicação através de uma rede ou localmente, de forma simultânea através de um mesmo endereço IP (Internet Protocol) e do protocolo TCP ou UDP (User Datagram Protocol), por exemplo, onde cada aplicação ou serviço possui sua porta específica. A porta padrão para acessar uma aplicação RIA baseada no uniGUI é a 8077, porém pode ser facilmente alterada através da propriedade "Port", também presente no objeto UniServerModule. Além do acesso a aplicação RIA, também é possível acessar um painel de controle da mesma através desta porta, simplesmente adicionando "/server".

Para dar continuidade ao exemplo, será criado um banco de dados utilizando o Firebird, que foi denominado "BASE.FDB", com uma simples tabela de produtos. A **Listagem 1** apresenta o código SQL utilizado na criação da tabela.

Após a criação da base de dados e da tabela produto, retornando ao Delphi, foram adicionados os objetos responsáveis pela ligação com os mesmos, ambos do Framework DBExpress. Sendo assim, foram adicionados ao objeto UniMainModule um TSQLConnection para a conexão, e um TSimpleDataSet para as operações com a tabela. Também foi inserido um TDataSource. As propriedades do SQLConnection1 receberam as definições para a conexão com o Firebird apresentadas na **Listagem 2**.

Nota do DevMan 5

O TCP (Transmission Control Protocol) é um dos protocolos que possibilita a transferência de informações entre as entidades envolvidas, através de uma rede, como é o caso da Internet. É muito empregado, pois, implementa mecanismos que asseguram a entrega da informação mantendo sua integridade, uma vez que a mesma trafega por meios inseguros, como é o caso da rede física.

Listagem 1. Código SQL da tabela Produto

```
CREATE TABLE produto (
  id INTEGER NOT NULL,
  descricao VARCHAR(100),
  valor NUMERIC(15,2),
  CONSTRAINT pky_produto PRIMARY KEY(id)
);
```

Listagem 2. Configurações com o Firebird no SQLConnection

```
ConnectionName = FBConnection
Params.Database = Base\BASE.FDB
LoginPrompt = False
Connected = True
```

Concluída a configuração do SQLConnectin1, o objeto TSimpleDataSet recebeu a denominação de "sdsProduto". Sua propriedade Connection foi definida com o valor "SQLConnection1" e seu comando SQL, atribuído em DataSet.CommandText, o valor "SELECT * FROM produto". Sua propriedade Active foi definida como True para ativar a conexão e foram adicionados os campos da tabela de produtos, através de um clique duplo sob o mesmo e da opção "Add all fields" presente no menu de contexto. Em seguida, para terminar a configuração do sdsProduto, seus eventos AfterDelete e AfterPost receberam a instrução "sdsProduto.ApplyUpdates(0)" para efetivar a devida gravação dos dados na tabela. Por fim, o objeto TDataSource recebeu a denominação de "dsProduto" e sua propriedade DataSet, o valor "sdsProduto".

Finalizada as configurações de conexão, a próxima etapa é elaborar a interface gráfica do Software Desktop e, consequentemente, da aplicação RIA. Neste momento, devem ser empregados os componentes que pertencem aos grupos presentes nas paletas do uniGUI, conforme a necessidade do projeto.

Portanto, foi adicionado ao formulário principal da aplicação (MainForm), um objeto do tipo TUniMainMenu para a elaboração de um simples menu. Este componente tem seu funcionamento de forma semelhante ao TMainMenu do Delphi, onde um simples clique duplo permite a elaboração de toda a estrutura de itens de menu e submenus. Sendo assim, foi criado um novo item de menu, denominado "Manutenção", com um subitem denominado "Produto". Para criar um novo formulário, responsável pela manutenção dos produtos, o menu File>New>Other deve ser acessado novamente. No grupo "uniGUI for Delphi", basta selecionar a opção "Form", e após clicar no botão OK, a opção de criação, "Application Form", pode ser mantida. Este novo formulário recebeu a denominação de "unifoProduto". Posteriormente, para elaborar a interface gráfica do mesmo, foram adicionados um TUniGroupBox, um TUniDBNavigator, um TUniButton, um TUniDBGrid, três TUniLabels e três TUniDBEdits. Também foi incluída a Unit MainModule na cláusula Uses do mesmo para possibilitar o acesso aos objetos de conexão. Na propriedade DataSource de todos os componentes de ligação para os campos e registros da tabela, tal como o TUniDBNavigador e TUniDBGrid, foi definido o valor "UniMainModule.dsProdutos", o DataSource adicionado e configurado anteriormente. Para o botão adicionado, foi inserida uma simples mensagem para a apresentação do Software. Para que a mesma possa ser exibida na aplicação RIA, basta usar um dos métodos de mensagem do Delphi, MessageDlg ou ShowMessage. Sendo assim, o evento OnClick deste botão, recebeu o método MessageDlg de acordo com o trecho a seguir:

```
MessageDlg('Bem vindo a aplicação RIA!',mtInformation,[mbOK]);
```

Terminada a configuração do unifoProduto e seus objetos, basta retornar ao formulário principal, MainForm, e definir o código para o evento OnClick do item de menu adicionado, denominado Produto, para que o mesmo exiba o formulário de manutenção de produtos quando o usuário clicar, lembrando que a Unit referente ao unifoProduto deve ser incluída na cláusula Uses. Sendo assim, seu evento OnClick recebeu o simples código para exibir o formulário::

```
unifoProduto.ShowModal;
```

Assim como este método é utilizado para exibir um formulário no Software Desktop, na aplicação RIA baseada no uniGUI, o mesmo também pode ser empregado para obter esta funcionalidade.

Neste momento, o exemplo está concluído. Para acessar a aplicação RIA, basta compilar e executar o Software. Em seguida, um navegador deve ser aberto e na barra de endereços, deve ser informado o URL (Uniform Resource Locator – **Nota do DevMan 6**) de acordo com a estrutura "http://<Endereço da máquina>:<Número da porta>", em outras palavras, "http://localhost:8077", onde localhost é o endereço de loopback da máquina, o computador local, e evidentemente, quando a aplicação for acessada através da rede, basta substituir localhost pelo endereço IP da mesma. A **Figura 4** e a **Figura 5** apresentam a aplicação RIA em execução através de um navegador.

Nota do DevMan 6

O URL (Uniform Resource Locator) é uma funcionalidade empregada para a localização e acesso rápido de determinado recurso através da Internet ou rede, possibilitando que aplicações externas possam encontrar e consumir facilmente o recurso em questão. O URL é composto basicamente pela definição do protocolo que deve ser utilizado, tal como o HTTP, e uma cadeia de caracteres (string), assim como é apresentado no exemplo "http://localhost".

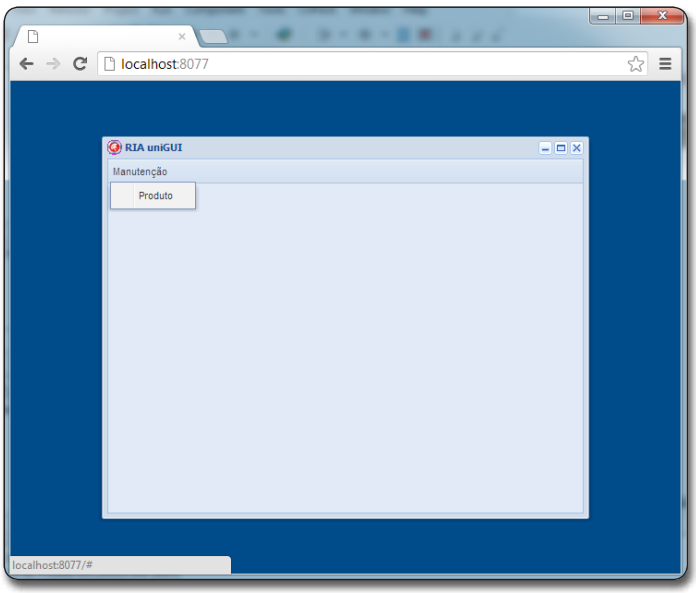


Figura 4. Página principal da Aplicação RIA uniGUI através de um navegador

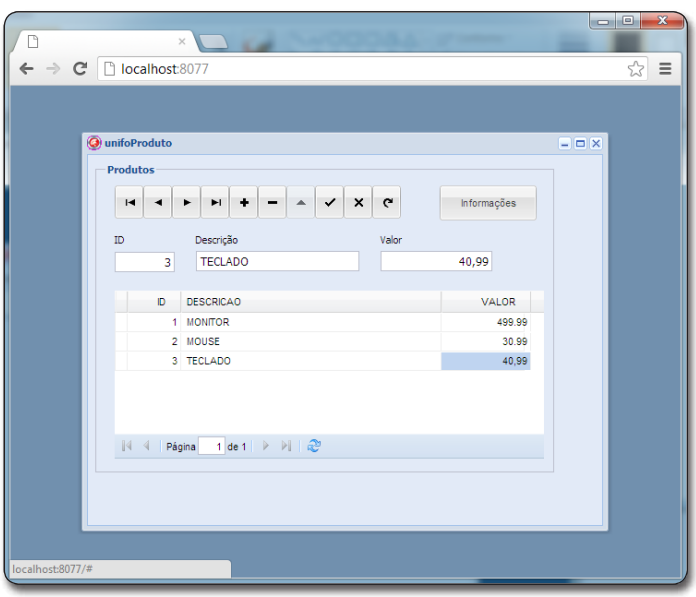


Figura 5. Página de manutenção de produtos da Aplicação RIA uniGUI

Como foi observado, este é um simples exemplo de como empregar o uniGUI para o desenvolvimento de aplicações RIA. Vale à pena ressaltar que o uniGUI vai muito além do que foi apresentado, possibilitando a elaboração de complexas aplicações que podem ser utilizadas para atender a área comercial e assim por diante.

Conclusão

É inegável a importância da Internet nos dias atuais não só para as pessoas, mas também para as organizações. Contudo, elaborar aplicações para Web, de início, requer da equipe de desenvolvimento, um conhecimento sólido sobre HTML, CSS, Javascript, e pelo menos uma linguagem server-side, como o PHP, bem como dominar o funcionamento do servidor HTTP e todos os recursos que envolvem uma aplicação Web que será utilizada simultaneamente por diversos usuários, como é o caso de sessões, cookies, etc. Contudo, para minimizar este trabalho muitas vezes desafiador, foram surgindo diversas ferramentas e Frameworks com o decorrer do tempo que possibilitaram muitas vantagens além da mencionada, abstraindo muitos conceitos e recursos que seriam necessários para que este objetivo fosse alcançado.

Um excelente Framework para o desenvolvimento RIA baseado em JavaScript, conforme foi mencionado, é o ExtJS, que por sua vez é a base para o funcionamento do conjunto de ferramentas e componentes de outro Framework, o uniGUI, o recurso que permite desenvolver RIA através do Delphi como se estivesse elaborando uma aplicação Desktop corriqueira com banco de dados, conforme foi demonstrando através de um exemplo básico. Além do mais, o uniGUI possibilitou que o desenvolvimento de aplicações RIA fosse efetuado sem qualquer domínio da linguagem HTML e JavaScript, embora evidentemente tal conhecimento sempre é bem vindo quando o assunto é a Web, bem como não houve a necessidade de elaborar muito código fonte para que a aplicação RIA estivesse pronta.

Sendo assim, o uniGUI é um grande aliado e um conjunto de componentes que realmente vale a pena ter a disposição, afinal, traz diversas vantagens para o desenvolvimento de RIA e ainda permite a elaboração de uma aplicação para Web sem o domínio de outras tecnologias senão o próprio Delphi.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Maiores informações e Download do uniGUI
http://www.unigui.com/

Maiores informações e Download do Raudus
http://www.raudus.com/

Maiores informações e Download do ExtJS
http://www.sencha.com/products/extjs/

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Componente cxGrid da DevExpress

Conheça um dos mais completos Grids do mercado

Em se tratando de informações, o Delphi sempre contou com ótimos componentes na VCL para visualização e manipulação de dados de uma forma interessante, e mesmo quando se faz necessário adicionar um recurso diferente, a customização é uma tarefa bem simples. Entre os componentes que tratam a visualização de dados podemos destacar o DBGrid que apresenta o formato linhas e colunas e o DBCtrlGrid, que permite uma visualização sofisticada a partir de um container de outros componentes (como Labels, Edits, DBTexts, DBEdits entre outros). No entanto, com o avanço tecnológico ocorrendo cada vez de forma mais agressiva, os usuários naturalmente estão se tornando cada vez mais exigentes, conhecendo novos recursos, exigindo novidades e isto é claro, faz com que cada vez haja a necessidade de se criar novas maneiras de atendam este grupo de usuários "avançados" em particular.

Um grande exemplo desta característica pode ser notada claramente nos níveis hierárquicos mais altos onde um gerente administrativo ou diretor passa horas analisando informações por meio de relatórios, gráficos, sempre com grandes quantidades de dados em tela, consultas avançadas/detalhadas, recursos de BI (**Nota do DevMan 1**) entre uma série de outros exemplos. Com isso, dependendo da situação (e do que o seu usuário deseja), nem sempre adequar seu Software a estes requisitos será uma tarefa fácil.

Pensando em detalhes como estes, o objetivo deste artigo é demonstrar através de uma forma prática, o quão simples pode ser utilizar impressionantes recursos visuais para agregar valor à sua aplicação e conquistar o seu cliente com um nível de informações extremamente rico e customizável. Sendo assim, apresentaremos neste artigo o cxGrid da suíte de componentes da DevExpress.

Durante o desenvolvimento de nosso exemplo, veremos como esta tarefa de customização dispensa codificações, fazendo que todo o processo concentre-se principalmente em configurações da Grid e de recursos

Resumo DevMan

Componente cxGrid da DevExpress:

Exibir um grande volume de dados de forma organizada visualmente pode não ser uma tarefa tão simples, o que pode fazer com que haja a necessidade de inserir vários componentes ou criar recursos mais complexos, na qual exige um trabalho maior para manter a tela limpa e organizada. O objetivo do artigo é ilustrar o componente cxGrid da suíte DevExpress, demonstrando o quão simples pode ser criar telas extremamente ricas em detalhes visuais, exigindo apenas configurações.

Em que situação o tema é útil:

O tema é útil na elaboração de recursos de consultas avançadas, permitindo a criação de telas complexas com grande quantidade de informações, já que a suíte DevExpress conta com uma gama de recursos interessantes e autoconfiguráveis, minimizando a necessidade de codificação por parte do desenvolvedor. Agregar recursos em sistemas visto que os componentes podem ser aplicados tanto em aplicações prontas quanto em desenvolvimento.

Nota do DevMan 1

BI (Business Intelligence) ou inteligência de negócios, como o próprio nome sugere, refere-se a um processo de tratamento de informações direcionadas ao ambiente de negócios. Com isso, são realizadas estratégias de coleta e organização dos dados para a obtenção de elementos de vantagem competitiva de mercado. Alguns casos de obtenção de informação estratégica que é resultante do BI pode ser, por exemplo, a quantidade de vendas de um determinado tipo de produto em uma determinada região, com isso, pode-se avaliar se este produto está sendo vendido da forma correta naquela região escolhida, qual seu público alvo e até mesmo questões que vão além do desenvolvimento, como estratégias de marketing complementar.

Existem ferramentas exclusivas para a realização desta tarefa, além de recursos de bancos de dados que dispõem destas características (como o Oracle, por exemplo). Entre elas podemos citar os famosos cubos de decisão (componente do Delphi), FastCube (componente produzido pela empresa que criou o gerador de relatório FastReport), ou mesmo o próprio cxGrid, componente apresentado neste artigo, que possui uma série de recursos estratégicos.

à parte. Abordaremos as partes de configuração, gráficos, impressão dinâmica e automática, agrupamento, ordenação, estilo, seleção de colunas entre outras.

Porque utilizar a suíte DevExpress?

Ao utilizar os componentes da suíte, pode-se notar uma grande diferença em relação aos componentes nativos da VCL. O grande diferencial da suíte neste caso é sua quantidade de recursos, que permite a construção de aplicações de porte mais simples até as mais complexas. A DevExpress detentora da suíte, disponibiliza estes componentes desde as versões mais antigas do Delphi, deste modo, é possível realizar praticamente todos os recursos vistos aqui no seu Delphi 7, por exemplo. Vale lembrar que a suíte de componentes da DevExpress, diversas vezes ganhou prêmios como melhor suíte de componentes, como pode ser visto na própria página da empresa.

Além do Delphi, ainda contam com o desenvolvimento de componentes para .NET, neste caso, englobando as tecnologias ASP .NET e Silverlight, valendo a pena conferir.

Caso você tenha interesse em adquirir apenas o cxGrid e não toda a suíte completa, é possível comprar apenas os componentes que você desejar.

Suítes de componentes similares

Embora não sejam exatamente iguais e nem possuam todos os recursos da suíte de componentes da DevExpress, podemos destacar várias outras suítes de componentes que podem dar uma "turbinada" no desenvolvimento Delphi. Entre elas podemos destacar a antiga RxLib (compatível até hoje com as versões mais novas), a suíte JEDI (substituta do RxLib) e a suíte de componentes da TMS, que possui recursos extremamente interessantes. Vale lembrar que a RxLib e a JEDI são projetos OpenSource, enquanto as bibliotecas DevExpress e TMS são pagas.

Tutorial

Instalando a suíte de componentes

O processo de instalação dos componentes é extremamente simples e no site do fabricante é possível baixar cada um dos controles de forma separada, permitindo assim testá-los de uma forma direcionada à sua necessidade.

Conhecendo a cxGrid

Pelo contrário do que a maioria pensa sobre a Quantum Grid (cxGrid), como sendo apenas uma grid comum com alguns recursos especiais, como a ordenação, filtro, agrupamento e cálculos (de colunas), o componente tem muito mais a demonstrar, indo além destas características.

Primeiramente, vale ressaltar que o componente cxGrid não é um Grid em si, mas sim um Container de Grids, no qual lhe permite exibir e/ou agrupar registros visualmente, exibindo-os de

Figura 1. CxGrid ao ser adicionada ao formulário

forma mestre-detalhe em um só, sem a necessidade de adicionar várias outras Grids para as demais informações. Adicionando o componente, inicialmente ele será exibido no formulário como mostrado na **Figura 1**.

A marcação da **Figura 1** corresponde ao mínimo que o Grid deve ter para ser exibido de forma básica. Neste caso as marcações referem-se às seguintes características:

- cxGrid1: refere-se ao nome do container.
- cxGrid1Level1: refere-se ao nível que será exibido à grade.
- cxGrid1DBTableView1: Este é o denotador utilizado para acessar as propriedades do Grid, como armazenamento de colunas, ligação com o DataSource e outras tantas, responsáveis pelos seus "efeitos especiais". Algumas destas propriedades são ilustradas na **Figura 2**.

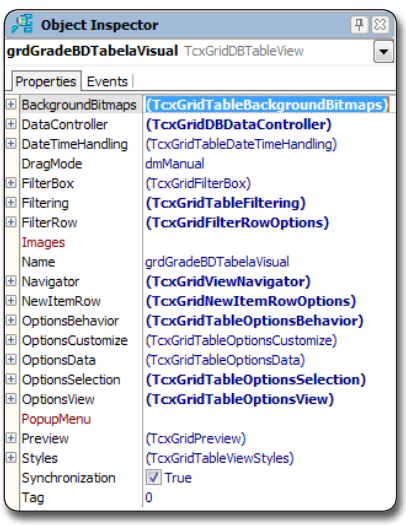


Figura 2. Propriedades do cxGrid1DBTableView1

A **Tabela 1** descreve algumas funcionalidades destas propriedades na **Figura 2**, bem como suas respectivas funcionalidades.

Após algumas explicações sobre as principais propriedades da Grid, veremos em pequeno exemplo de projeto feito para demonstrar o uso do componente, criando uma pequena aplicação utilizando os recursos para exibir dados.

Criando o banco de dados

Nosso exemplo ilustra uma busca de informações. Foi utilizado o banco de dados Firebird 2.5, mas os recursos aqui vistos são independentes de banco de dados. Foi criado um banco de dados e nele, uma tabela chamada FUNCIONARIOS (**Listagem 1**) foi adicionada. Para efeito de demonstração, essa tabela foi populada manualmente.

Listagem 1. Tabela de funcionários

```
CREATE TABLE FUNCIONARIOS(
  IDFUNCIONARIO INTEGER NOT NULL PRIMARY KEY,
  NOME VARCHAR(250) NOT NULL,
  DATA_NASCIMENTO DATE,
  DATA_ADMISSAO DATE,
  ENDERECO VARCHAR(250),
  NUMERO INTEGER,
  BAIRRO VARCHAR(250),
  CEP VARCHAR(8),
  CIDADE VARCHAR(250),
  ESTADO CHAR(2),
  SALARIO DOUBLE PRECISION,
  STATUS CHAR(1) NOT NULL,
  OBSERVACAO BLOB SUB_TYPE 1 SEGMENT SIZE 80,
  FOTO BLOB SUB_TYPE 0 SEGMENT SIZE 80
  );
```

Criando o projeto

Como o objetivo do nosso exemplo é simplificar a aplicação, demonstraremos como aproveitar ao máximo os recursos da Grid sem ter que escrever grandes códigos ou mesmo complexos. Sendo assim, criamos uma aplicação comum do tipo VCL Forms (File>New>VCL Forms Application – Delphi). Utilizamos o próprio formulário como base da aplicação, deste modo, não foram implementadas telas de cadastro ou mesmo outros recursos. O formulário principal foi renomeado para frmCadastroFuncionarios, sua unit foi salva com o nome de untCadastroFuncionarios.pas e o projeto como CadFuncionarios.dproj.

Já na primeira tela foi adicionado um componente SQLConnection (dbx) realizando o processo de conexão com o banco de dados criado anteriormente. Neste caso o componente foi nomeado para Conexao. Sua propriedade LoginPrompt foi marcada como False e selecionado em ConnectionName a conexão previamente criada. Em seguida foi adicionado ao formulário os componentes SQLDataSet (sqlFuncionarios), DataSetProvider (dspFuncionarios), ClientDataSet (cdsFuncionarios) e DataSource (dsrFuncionarios). O SQLDataSet é ligado ao componenente Conexao por meio da propriedade SQLConnection e em seguida, na propriedade CommandText é informad a sentença SQL utilizada para a busca dos campos no processo de consulta. A instrução completa encontra-se na **Listagem 2**.

Nome da propriedade	Funcionalidade
BackgroundBitmaps	Esta opção refere-se à personalização do Grid, utilizando imagens de fundo como texturas e outras figuras no formato “bmp”.
DataController	Refere-se à conectividade com o DataSet, permitindo configurações de carregamento de dados, pré-filtro, entre outras.
DateTimeHandling	Oferece várias propriedades que afetam a maneira como os valores de data e hora são filtrados e agrupados.
Filterbox	Está diretamente relacionada a vários aspectos da aparência da barra de filtro. É exibida quando é utilizado o recurso de filtro da própria Grid.
Filtering	A propriedade se refere ao recurso de filtro, que está disposto sobre as colunas, representado pelo ícone de um “funil”, posicionado no canto direito do título da coluna.
FilterRow	Responsável pela exibição de um painel fixo abaixo do título das colunas, como linha fixa, para serem informados dados para o filtro.
Navigator	Corresponde à propriedade para exibição de uma barra de navegação, se habilitada, a mesma é exibida no canto inferior esquerdo. Possui opções para exibir/ocultar botões.
NewItemRow	Responsável pela exibição de um painel fixo abaixo do título das colunas, como linha fixa, se não estiver setada para True a propriedade FilterRow, caso contrário, será exibida após a FilterRow, para ser usada como método de inclusão de novos registros.
OptionsBehavior	Propriedade responsável pelo comportamento, conforme ações do usuário, tal como o pressionar de teclas. Faz com que os editores de cada célula fiquem sempre ativos, permitindo com que o foco de cada célula possa ser alterado através das teclas Tab e/ou Enter entre outras.
OptionsCustomize	Determina as características que podem ser ativadas em tempo de execução como manipulações de colunas , por exemplo, permitindo ao usuário final mover, filtrar, ordenar, redimensionar, entre outras, ou não permitir nenhuma das ações citadas.
OptionsData	Determina quais tipos de ações poderão ser executadas no Grid, sendo elas, incluir, editar e excluir.
OptionsSelection	Esta opção determina como o usuário irá ocorrer à seleção dentro do Grid, podendo ser em forma de células individualmente, linhas inteiras, múltiplas linhas, entre outras.
OptionsView	Esta propriedade especifica a visibilidade de rodapés para exibição de colunas de resumo/agregação como: “Maior Valor”, “Menor Valor”, “Média”, “Soma” e Contar; Aqui vale a observação de que se a coluna possuir o formato de texto, somente será possível utilizar a agregação de contagem de registros. Também é possível ativar rodapés em modo de agrupamento, exibição de cabeçalhos, desativar a caixa de agrupamento (que por padrão já vem habilitada) podendo assim impedir que o usuário agrupe alguma informação, ou fixando este agrupamento em somente uma coluna.
Preview	Proporciona uma exibição privilegiada de uma coluna, onde a mesma será exibida na parte inferior (por padrão) ou superior a cada registro exibido dentro da Grid, ideal para colunas onde seu conteúdo seja um texto extenso, como uma observação por exemplo.
Styles	Refere-se à aparência da Grid, podendo alterar as cores e fontes das áreas da Grid, desde a caixa de agrupamento, o rodapé, linhas de registros, como a famosa Grid “zebrada”, através das propriedades “ContentEven” (registros pares) ou “ContentOdd” (registros impares).

Tabela 1. Propriedades do componente cxGrid

Listagem 2. SQL do componente sqlFuncionarios

```
SELECT
  FUNCIONARIOS.IDFUNCIONARIO,
  FUNCIONARIOS.NOME,
  FUNCIONARIOS.DATA_NASCIMENTO,
  FUNCIONARIOS.DATA_ADMISSAO,
  FUNCIONARIOS.ENDERECO,
  FUNCIONARIOS.NUMERO,
  FUNCIONARIOS.BAIRRO,
  FUNCIONARIOS.CEP,
  FUNCIONARIOS.CIDADE,
  FUNCIONARIOS.ESTADO,
  FUNCIONARIOS.SALARIO,
  FUNCIONARIOS.STATUS,
  FUNCIONARIOS.OBSERVACAO,
  FUNCIONARIOS.FOTO
FROM
  FUNCIONARIOS
```

Note que o SQL não possui uma condição **WHERE**, o que obviamente poderia trazer problemas de performance em uma aplicação maior já que todos os registros existentes dessa tabela seriam carregados, no entanto, utilizaremos este exemplo apenas para demonstrar os registros sendo exibidos, além de outros recursos da Grid, como anteriormente mencionado.

Após a inserção do SQL, selecione o componente dspFuncionarios e em sua propriedade DataSet aponte para o sqlFuncionarios. A seguir, no ClientDataSet (cdsFuncionarios) ligue a propriedade ProviderName no dspFuncionarios e por último, informe o cdsFuncionarios na propriedade DataSet do DataSource (dsrFuncionarios). Com isso, as respectivas ligações estão devidamente realizadas.

Para o desenvolvimento do Layout, insira um componente Panel (pnlBusca) com alinhamento (Align) como alTop. Dentro dele, adicione dois Buttons (btnAbrir e btnImprimir). Por último, adicione um cxGrid. Clique sobre ele e selecione o item principal que se refere ao container, na qual deverá aparecer inicialmente com o nome de cxGrid1 e configure seu Align como alClient, alterando seu nome para grdGrade.

Selecione o ClientDataSet e dê um duplo clique de esquerda sobre o mesmo para que o Fields Editor seja exibido. Clique sobre ele e selecione a opção "Add all fields" para que todos os campos sejam acrescentados ao cdsFuncionarios, facilitando assim a criação dos campos dentro da Grid, que veremos mais adiante.

Logo após a configuração do ClientDataSet, implemente o evento OnClick do botão Abrir conforme a **Listagem 3**.

O código é muito simples, apenas é realizado o fechamento da conexão (linha 03) e a abertura em seguida (linha 04), fazendo com que os dados sejam automaticamente carregados em memória no ClientDataSet e consequentemente, exibidos na Grid assim que conectada.

Após esse procedimento, adicionaremos o evento AfterPost no cdsFuncionarios, conforme a **Listagem 4**.

Esse se refere ao comportamento do ClientDataSet, que logo após receber uma instrução de Post (salvar o registro), irá executar o procedimento da linha 03, que refere-se a persistência no banco de dados. O parâmetro 0 dentro do método ApplyUpdates indica a quantidade de erros permitidos neste processo. Ao informar 0 (zero) determinamos que caso algum erro ocorra, nada seja salvo.

De forma exatamente igual codificamos o evento AfterDelete, para que, assim que for disparada uma exclusão, a mesma seja executada no banco de dados.

O resultado da tela deverá ser parecido com o da **Figura 3**.

Listagem 3. Evento OnClick do botão btnAbrir

```
01 procedure TfrmCadastroFuncionarios.btnAbrirClick(Sender: TObject);
02 begin
03   cdsFuncionarios.Close;
04   cdsFuncionarios.Open;
05 end;
```

Listagem 4. Evento AfterPost do cdsFuncionarios

```
01 procedure TfrmCadastroFuncionarios.cdsFuncionariosAfterPost(DataSet:
     TDataSet);
02 begin
03   cdsFuncionarios.ApplyUpdates(0);
04 end;
```

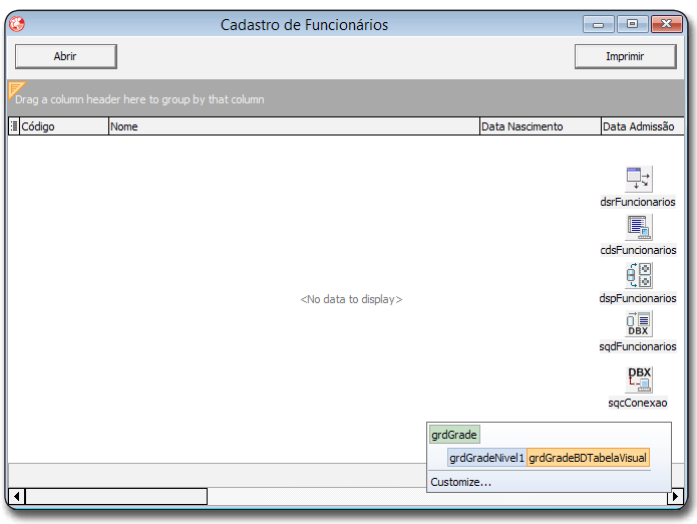


Figura 3. Layout proposto da aplicação

Após ter realizado as ligações e configurações conforme já mencionado, vamos ao processo de conexão do Grid com os dados. Diferentemente dos Data Controls do Delphi, a ligação dos dados não é realizada diretamente na propriedade DataSource. Neste caso, para conectarmos o Grid ao DataSource, clique sobre o grdGradeBDTabelaVisual e expanda no Object Inspector a propriedade DataController, conforme a **Figura 4**. Deste modo, basta conectar o DataSource à sua respectiva propriedade. Sendo assim, caso fosse necessário realizar uma ligação em Runtime, bastaria realizar o seguinte procedimento:

```
grdGradeBDTabelaVisual.DataController.DataSource := dsrFuncionarios;
```

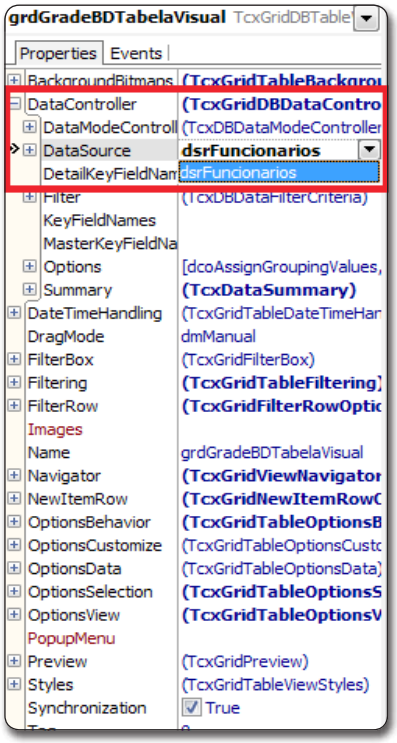


Figura 4. Configuração do DataSource dentro de DataController

Feito esse procedimento, a Grid e o ClientDataSet estão devidamente vinculados. Podemos então criar as colunas. Se você adicionou os campos ao FieldsEditor do ClientDataSet, esse processo poderá ser feito de forma automática, bastando para isso apenas um clique duplo sobre a área em branco do Grid para que a janela de configuração seja apresentada (**Figura 5**).

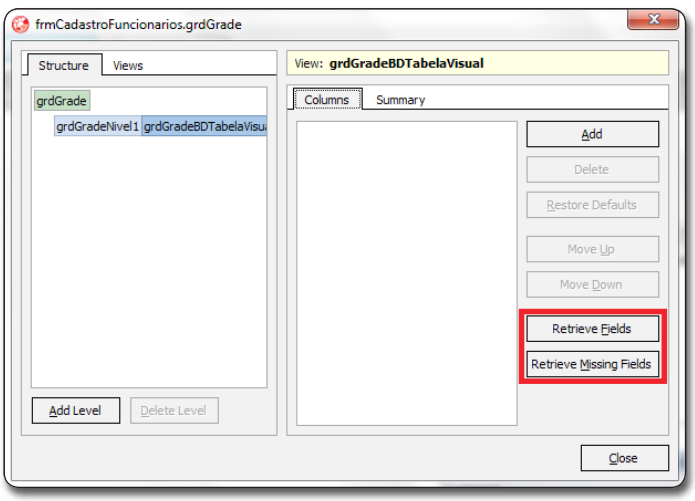


Figura 5. Tela de configuração de campos

Nela podemos configurar/gerenciar os campos dentro do Grid. Para criar as colunas automaticamente basta utilizar os botões em destaque na mesma figura, lembrando que para isso é necessário que o DataSet esteja ativo, ou a propriedade Fields do mesmo esteja com campos adicionados.

O botão "Retrieve Fields" cria automaticamente todas as colunas, se as mesmas já estiverem sido criadas, será apresentada uma mensagem questionando se você deseja excluir as colunas existentes (caso já existam), selecionada a opção "Sim", as colunas existentes serão excluídas e novas serão criadas baseadas nas propriedades dos campos no DataSet. Vale lembrar que qualquer configuração anteriormente realizada em qualquer coluna será perdida, selecionada a opção "Não", as colunas existentes serão mantidas e novas serão criadas (apenas as não existentes).

Já o botão "Retrieve Missing Fields" faz uma verificação se existe algum campo diferente no DataSet, que não esteja criado e adiciona somente a diferença (o mesmo quando é respondido "Não" à exclusão das colunas).

O resultado da janela após a criação das colunas será como mostrado na **Figura 6**.

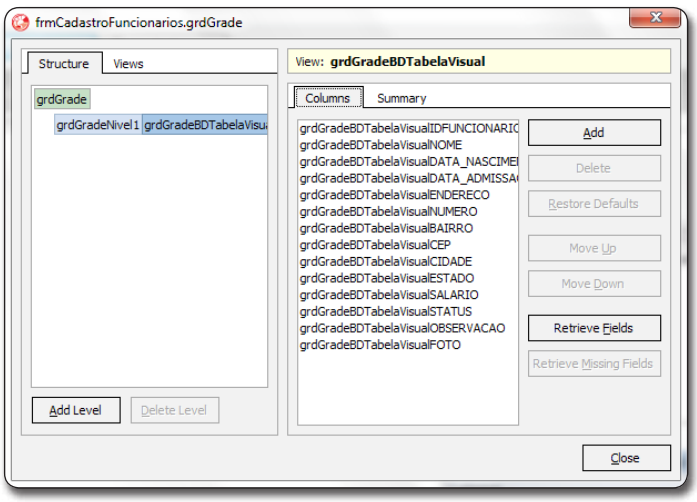


Figura 6. Colunas adicionadas ao Grid

Após esse processo, devemos personalizar as colunas modificando seus respectivos Captions. Vale lembrar que caso a propriedade "Display Label" dos Fields esteja preenchida, suas definições serão aplicadas aos Captions. No entanto, para fazer esse procedimento podemos selecionar a coluna diretamente a partir da janela (**Figura 6**) clicando sobre o campo da lista e fazendo com que suas propriedades sejam exibidas no Object Inspector, bastando apenas modificar o Caption da coluna selecionada. Outra forma é através da própria Grid que disponibiliza esse recurso, neste caso vamos optar por esta maneira, então a janela atual pode ser fechada. O Grid com as colunas criadas é mostrado na **Figura 7**.

Para customizar o Grid, primeiramente iremos ajustar o tamanho das colunas para que as mesmas sejam mais fáceis de selecionar, para isso, clique sobre a área em branco do Grid para que o menu da **Figura 8** seja exibido.

Clique sobre a opção "Edit Layout and Data...". Neste momento será apresentada uma janela que nos dará uma pré-visualização da Grid, ou seja, de como ficaria em Runtime. Em qualquer uma das colunas, clique de direita sobre o título da mesma e selecione a

propriedade "Best Fit (all columns)". Com isso, o Grid irá se "auto ajustar" para obter uma melhor visualização das colunas, clique no botão OK e repare que todas as colunas irão estar dispostas da mesma forma que no modo de pré-visualização.

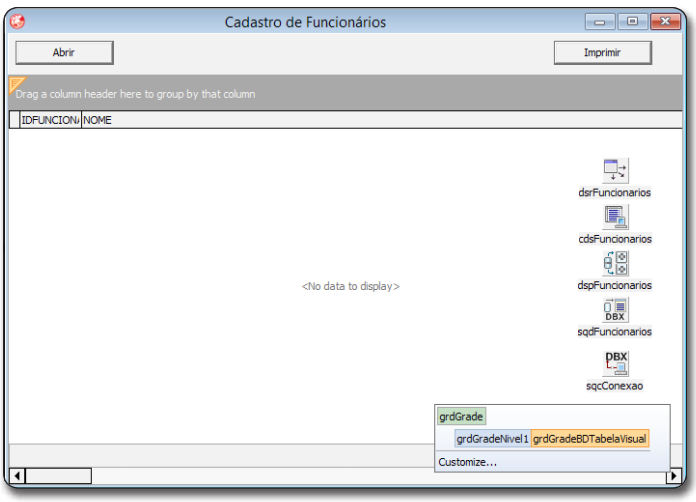


Figura 7. Adição de colunas no Grid

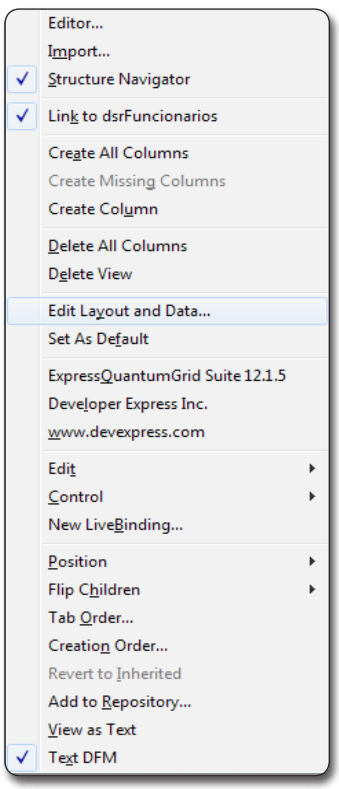


Figura 8. Menu de contexto para cuztomização da Grid

Para alterarmos as propriedades das colunas, podemos clicar diretamente sobre elas, como exibido na **Figura 9**.

Note que a coluna terá uma borda quando clicada e suas propriedades serão exibidas no Object Inspector, iremos trocar os Captions padrões que são recebidos do DataSet, para títulos mais amigáveis aos usuários.

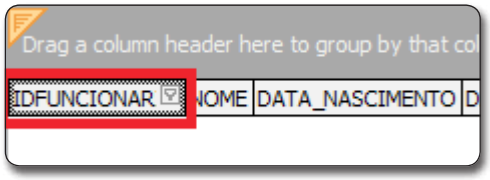


Figura 9. Customização da coluna com base no clique

Outro componente interessante é o cxGridPopupMenu (**Figura 10**). Para habilitar esse recurso, o componente TcxGridPopupMenu deve ser adicionado ao formulário e a sua propriedade Grid conectada ao cxGrid. Ele é responsável pela exibição dos menus que são exibidos clicando com o botão direito do mouse sobre os títulos das colunas (**Figura 11**) e sobre o rodapé (**Figura 12**).

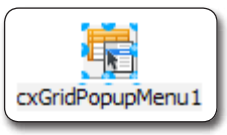


Figura 10. Componente cxGridPopupMenu

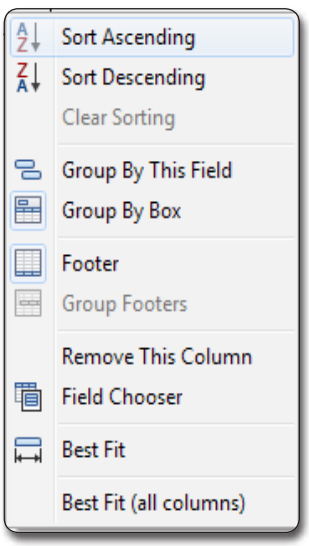


Figura 11. Menu de contexto relativo ao clique da coluna

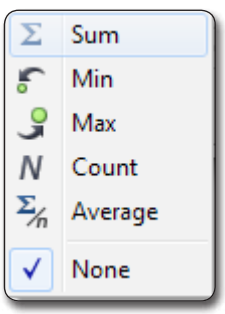


Figura 12. Menu de contexto relativo ao clique do rodapé

Exibido quando clicado sobre o título de qualquer coluna, o menu da **Figura 11** oferece opções de ordenação das colunas de forma crescente/decrescente, agrupamento, exibição do rodapé, redimensionamento da coluna ou auto ajuste de todas as colunas,

realizado de forma automática, efeito bem parecido com o de algumas ferramentas de edição de planilhas, como o Excel por exemplo.

O segundo menu (**Figura 12**) é responsável pelas funções de resumo que aparecem no rodapé, alinhado a cada coluna. Através dele é permitido cálculo de colunas numéricas ou contagem de registros em colunas de texto.

Para que o rodapé seja exibido, acesse o grdGradeDBTabelaVisual e vá até o conjunto de propriedades OptionsView do Grid, localize a sub-propriedade Footer e habilite-a para True.

Após esses procedimentos citados, já teremos basicamente os principais e mais cobiçados recursos da QuantumGrid. Ainda é possível realizar mais customizações, tornando-o extremamente diferenciado de um componente comum da VCL. O que faremos agora é incluir o recurso de cadastro por meio da própria Grid, permitindo incluir, excluir e editar dados.

Clicando sobre o Grid grdGradeBDTabelaVisual, em meio aos conjuntos de propriedades, existe uma que se chama Navigator, que refere-se à exibição de uma barra de navegação (semelhante ao dbNavigator comum), após expandi-la, podemos observar suas propriedades, dentre elas encontram-se Buttons, InfoPanele Visible. Fazendo um breve apanhado sobre essas propriedades temos:

• **Buttons**: Refere-se aos botões que compõem a barra de navegação de modo semelhante ao Navigator, como já mencionado. Entre eles temos os botões relativos a navegação (primeiro, anterior, próximo e último), os de manutenção (inserir, editar, excluir, cancelar e salvar) e por fim os Bookmarks (marcadores) e filtros que são exclusivos desta barra de navegação. Expandindo da mesma, podemos observar todos os botões da barra, dentro de cada conjunto, existe a possibilidade de controlar vários comportamentos dos botões, como Enabled, Hint e ImageIndex para personalização dos botões com imagens, previamente colocadas em uma lista e por fim, Visible, que é autoexplicativa.

• **InfoPanel**: Se trata da exibição de um painel no meio da barra, informando qual registro está selecionado e quantos existem.

• **Visible**: Propriedade relacionada à exibição da barra de navegação, no qual quando ativa, a navegação é exibida na parte inferior esquerda do Grid.

Neste exemplo estaremos usando a barra de navegação, então vá até a propriedade Visible do Navigator e configure-a com o valor True, o resultado será como a da **Figura 13**.

Após essas e outras configurações, você já poderá efetuar operações de manipulação com os registros, podendo incluir, editar e excluir através da barra de navegação exibida, como na área em destaque da **Figura 13**.

Por padrão o cxGrid não exibe o indicador do lado esquerdo que mostra uma espécie de seta/flecha, responsável por indicar a posição do cursor dentro da Grid, se achar necessária sua exibição, como no formato visual do DBGrid comum, selecione grdGradeBDTabelaVisual vá até o conjunto de propriedades OptionsView e localize a propriedade Indicatore deixando-a com o valor True.

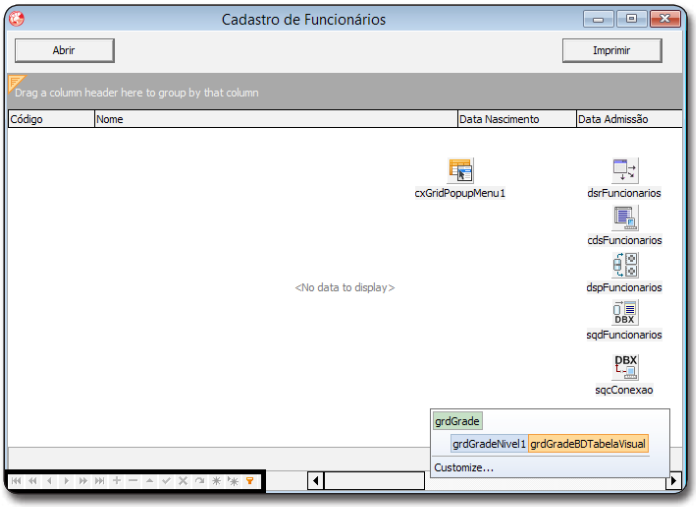


Figura 13. Navigator da Grid habilitado

Outras funcionalidades interessantes ficam no conjunto de propriedades denominado OptionsBehavior, são elas:

• **FocusCellOnTab**: Permite navegar entre as células através da tecla Tab.

• **FocusCellOnCycle**: Essa propriedade, estabelece um ciclo entre as células, onde caso ocorra a ação de passar de uma célula para outra utilizando a tecla Enter, a combinação entre as teclas Shift+Enter, ou mesmo as setas para esquerda e direita, sendo detectada a última coluna, automaticamente passará para o próximo registro posicionando o foco na primeira coluna, facilitando assim a navegação entre as células.

• **GoToNextCellOnEnter**: Quando ativa, permite ao usuário passar de uma célula para outra, apenas pressionando a tecla Enter, sem a necessidade nenhuma da intervenção do desenvolvedor com qualquer tipo de código.

• **FocusFirstCellOnNewRecord**: Propriedade muito interessante, quando ativa verifica se o foco está na última coluna e no último registro, se as duas condições forem correspondidas é criado um novo registro com o foco na primeira coluna, deixando o registro pronto para ser preenchido.

Podemos também privar o usuário de executar a operação de edição de dados, por exemplo, se quisermos um Grid que seja somente leitura, através do conjunto de propriedades OptionsData, isso é possível, ativando ou não as propriedades mais comuns, como Inserting, Editing, Deleting entre outras.

Agora veremos outra qualidade do Grid, que é a implementação de outros componentes dentro de qualquer coluna. Isto permite uma customização, onde cada registro pode ser um CheckBox, DBLookupComboBox, ComboBox, Memo entre tantos outros, isso sem nenhuma linha de código.

Por exemplo, no projeto, dentro da tabela FUNCIONARIOS existem três campos criados especificamente para ilustrar esse recurso, um destinado a armazenar o Status do funcionário, que será configurado para ser exibido na forma de CheckBox, outro

campo é a Observação, que será um BlobEdit e por último, o campo Foto que exibirá a imagem diretamente do registro.

Para começar, selecione a coluna Status, usando a rolagem horizontal para localizá-la no Grid (ou com um clique duplo do mouse com o botão esquerdo sobre a área branca do Grid), assim será exibido o Editor. Depois de selecionada a coluna é possível notar no Object Inspector as propriedades da mesma, logo abaixo haverá a propriedade denominada Properties, na lista a ser exibida localize a opção CheckBox selecione-a, repare neste instante após a seleção que a propriedade se transformará em um "subconjunto", expandindo esse subconjunto é possível observar que estão dispostas as propriedades relacionadas ao componente CheckBox. Para que o recurso funcione corretamente devemos configurar ValueChecked para uma String, que nesse caso é definida como "S" (sem aspas) e o atributo ValueUnchecked, que será "N" (também sem aspas).

Após esse processo configuraremos a coluna Observação utilizando o mesmo procedimento usado na coluna anterior, só que dessa vez iremos através do atributo Properties selecionar a opção BlobEdit dentro da lista, alterando a propriedade BlobEditKind para bekMemo.

Por fim, customizando a coluna foto, configure o atributo Properties, selecionando a opção Image, altere a propriedade GraphicClassName para TJPEGImage ou para o formato que deseja salvar no banco e ative a propriedade Stretch. Com isso, conseguimos facilitar a edição de dados diretamente no Grid que agora possui uma coluna como CheckBox, uma como BlobEdit e uma como Image.

A única peculiaridade é no campo Imagem, que para o usuário final incluir uma imagem, deverá clicar com o botão direito do mouse sobre a célula da Imagem, e selecionar o item Load, apontando na o caminho em que se encontra a figura que deseja. Uma boa alteração para o usuário final é permitir uma forma mais simples de visualizar as imagens, deixando a propriedade CellAutoHeight ativa (dentro do conjunto OptionsView), assim teremos um resultado como mostrado na **Figura 14.**

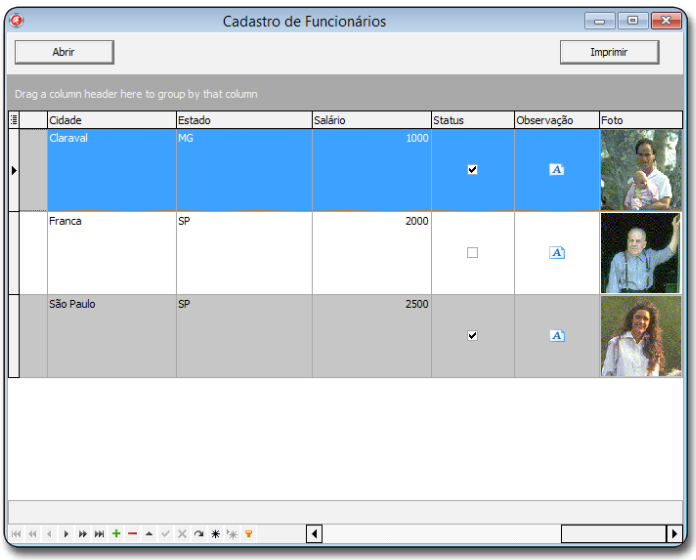


Figura 14. Propriedade CellAutoHeight ativada

Multiplos níveis

Essa característica é particular desta Grid, como citado anteriormente, o cxGrid refere-se a um Container, ou seja, podemos alojar diversos outros Grid, ou objetos nativos do QuantumGrid como, por exemplo, o cxGridDBChartSeries. Para adicionarmos outros objetos ao Grid, basta ralizar um duplo clique sobre sua área em branco e automaticamente será exibido o Editor, como vimos anteriormente na **Figura 6**.

Vale observar que ao lado da aba Structure, existe a aba Views. Selecione a mesma e dentro dela há uma lista que contém os objetos disponíveis para o QuantumGrid. Atráves dessa lista é possível tanto excluir o objeto criado por padrão como acrescentar novos.

Logo abaixo há o botão "Add View...". Clicando sobre ele será exibido um menu, nele estão contidos os tipos de objetos da QuantumGrid, na qual todos possuem o mesmo objetivo, exibir dados de um DataSet, com a diferença na qual são dispostas estas informações e a forma de navegação entre elas. Utilizaremos aqui no nosso exemplo o DB Chart para ilustrar essa funcionalidade. Selecionando a opção DB Chart, note que outro objeto será adicionado à lista, este objeto recebeu o nome de grdGradeDBChartView. Assim como o processo de conexão com o Grid, ligaremos este DBChart ao dsrFuncionarios através da propriedade DataControler>DataSource. Em seguida selecione a opção ToolBox e habilite a propriedade DiagramSelector, no qual permitirá a escolha entre diversos formatos de gráficos, como gráficos de barra, pizza, área, linha entre outros.

Ainda no Editor, do lado esquerdo existem os botões de manutenção Add, Delete e outros, clique sobre a aba Series e em seguida sobre o botão "Add...", serão exibidos somente campos numéricos, pois nesse caso a operação é referente à soma, escolha a coluna "SALARIO", com o item da lista Series selecionado, vá até a sua propriedade DisplayText e configure para Salário, de forma que fique mais amigável.

Até então, esse novo objeto não será exibido e o que irá determinar a sua exibição é o nível. Para isso teremos que acrescentar um novo nível, nesse caso para exibir mais de um item no container.

Voltando a aba Structure do Editor, observe que existe um nível. Por padrão ele é automaticamente criado e está ligado a outro objeto, para ver esse ligação clique sobre o mesmo e visualize o Object Inpector, através da propriedade GridView.

Para exibirmos outro objeto deve-se criar outro nível, para isso, clique sobre o grdGrade (container) selecionando-o, e em seguida escolha a opção Add Level logo abaixo. Após isto, outro nível será adicionado na mesma hierarquia do nível anterior, o mesmo sistema usado em objetos Treeview, por exemplo.

Agora com o novo nível criado, selecione-o, e em sua propriedade GridView, escolha o objeto grdGradeDBChartView, até então, ele ainda não será exibido, somente o primeiro nível/objeto será exibido. Devemos então selecionar o primeiro nível e configurar

sua propriedade Caption para "Funcionários", modificando também a propriedade DetailTabsPosition (dentro de Options) para o valor "dtpTop". Repita o processo com o segundo, porém, desta vez informe o texto "Gráfico" no Caption.

Finalizadas as configurações podemos fechar o Editor. Para concluir, selecione o container grdGrade principal. Este possui um conjunto de propriedades RootLevelOptions, apenas configure sua propriedade DetailTabsPosition para o valor "dtpTop" e pronto. Analisando o resultado repare que no topo da Grid existem duas abas, ou seja, cada um dos níveis agora são exibidos como abas, isso tudo somente utilizando o QuantumGrid, sem nenhum tipo de codificação, apenas com configurações. Observe o resultado das abas na **Figura 15** bem como a visualização dos gráficos (**Figura 16**).

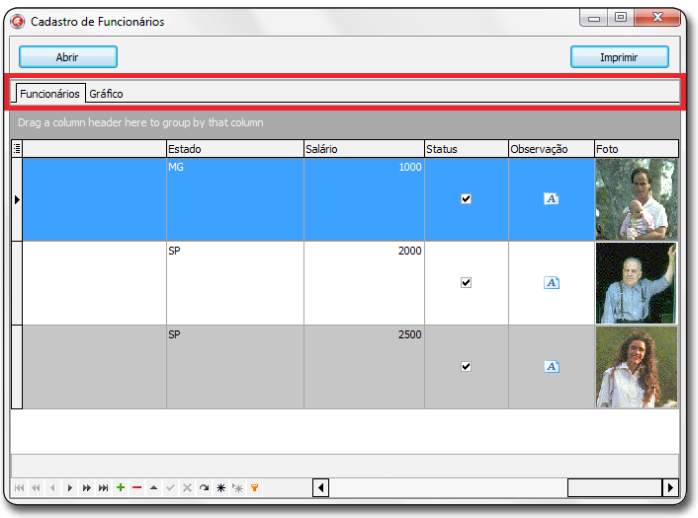


Figura 15. Grid em abas

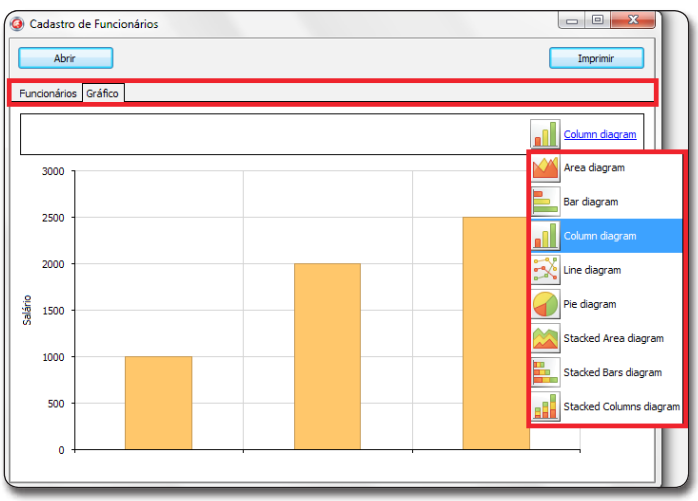


Figura 16. Aba de gráficos dentro do grid

Customizando o visual

Uma outra facilidade que o Grid disponibiliza, é o conjunto Styles, que serve para a personalização do modo como é apresentado. Sendo assim é possível personalizar as fontes e cores do Grid, realizando os efeitos de "Grid Zebrada", por exemplo, de uma forma fácil e rápida, através da propriedade ContentEven, clicando sobre a mesma e selecionando a opção "<Create a new style in the new repository>". Com isto o Grid adicionará automaticamente um novo componente chamado cxStyleRepository1. Este componente pode armazenar conjuntos de estilos, na qual pode ser configurado para conter informações de cor, fonte, cor do texto e imagem para textura.

Esse componente pode ser muito útil se você gosta de definir padrões para controlar formatações, possibilitando centralizar inúmeras customizações dentro de si, fazendo com que seja fácil aplicá-la a outros componentes visuais presentes na mesma tela. Para isto, basta selecionar o estilo que desejar, através da propriedade Style. Com duplo clique sobre o componente será exibida uma janela, dentro da mesma está contida uma listagem. Note que já há um estilo criado automaticamente pelo Grid com o nome de cxStyle1, altere somente a propriedade Color e confira o resultado na aplicação.

Impressão – Printing System

Fazendo parte da suíte de componentes da DevExpress, existe um conjunto chamado Printing System. Esses componentes são responsáveis pela impressão e navegação do conteúdo de alguns componentes da suite e em alguns casos até de outros que não fazem parte da suite, sendo caracterizados por possuir fácil configuração. Para utilizarmos este modo basta adicionar o componente dxComponentPrinter. Para configurá-lo, dê um duplo sobre o mesmo para que a janela de configuração seja exibida. Nela clique sobre o botão "Add..." para que outra janela seja exibida, como a ilustrada na **Figura 17**.

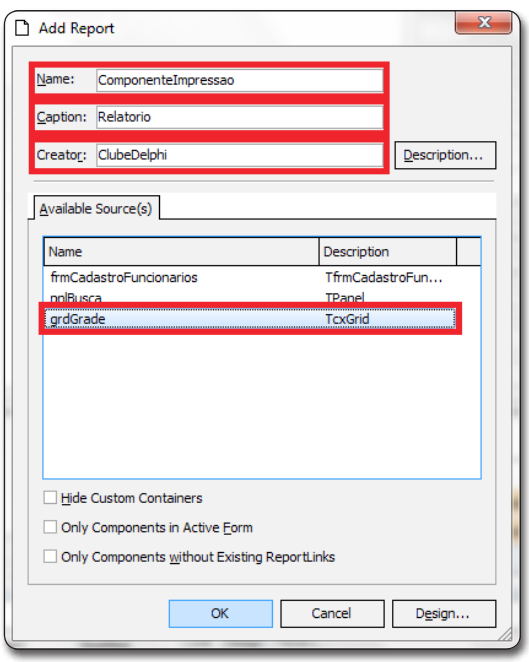


Figura 17. Configuração do componente de impressão

Na parte superior existe o campo Name. Altere o seu conteúdo para "ComponenteImpressao" e substitua o Caption por "Relatório". Dentro do editor há uma lista de componentes "Available Source(s)", localize o componente Grid e clique em OK para que a conexão seja definida. A pré-visualização/impressão corresponderá à aba selecionada do Grid. Feito este procedimento devemos acrescentar o código responsável por chamar a janela de pré-visualização e impressão do conteúdo, para isso, configure o evento OnClick do botão btnImprimir conforme a **Listagem 5**.

Listagem 5. Impressão do preview no evento onClick do botão

```
01 procedure TfrmCadastroFuncionarios.btnImprimirClick(Sender: TObject);
02 begin
03   ComponenteImpressao.Preview;
04 end;
```

Esse refere-se à chamada da pré-visualização dos dados, antes de fazer a impressão. Para deixar a impressão ainda mais interessante pode-se habilitar um recurso na Grid de personalização de colunas, no qual o usuário pode selecionar facilmente qual coluna exibir para impressão, para isso, selecione a propriedade OptionsCustomize e ative a subpropriedade "ColumnsQuickCustomization" para True, dessa forma o Grid tera a seleção de colunas como na **Figura 18.**

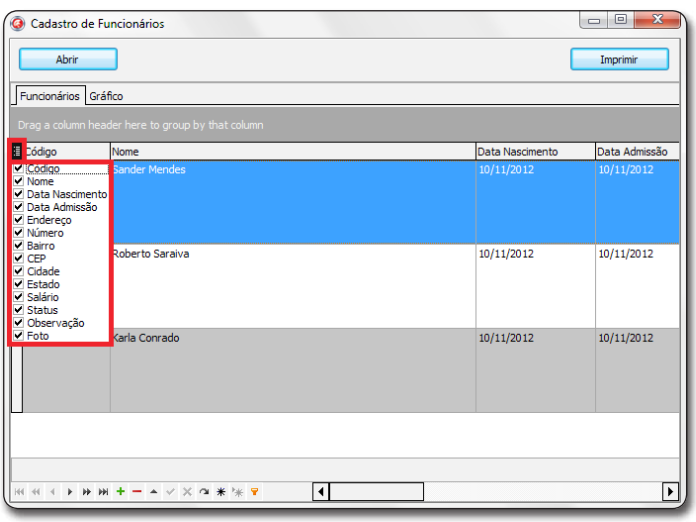


Figura 18. Seleção de colunas para impressão no relatório

Com isso, torna-se mais fácil configurar a impressão, pois será possível visualizar somente as colunas desejadas. A impressão será realizada de acordo com a aba escolhida. Na **Figura 19** é possível ver a impressão final após a customização e seleção da aba de gráficos.

Conclusão

O objetivo do artigo foi o de demonstrar o quanto é fácil criar aplicações com recursos visuais diferenciados sem que para isso seja necessário desprender um grande esforço. Utilizar o componente cxGrid pode fazer com que você tenha um aumento significativo na parte visual e no tratamento das informações. Como se percebe, a maioria dos recursos dispensa codificação, exigindo apenas a compreensão das propriedades e seu respectivo funcionamento, fazendo assim com que o trabalho de customização seja simples para o desenvolvedor, e consequentemente, permitindo ao usuário uma flexibilidade maior para lidar com grandes volumes de dados. Embora tenhamos visto vários dos principais recursos, ainda há muito a ser mostrado tanto em relação a este componente quanto da suíte DevExpress em si.

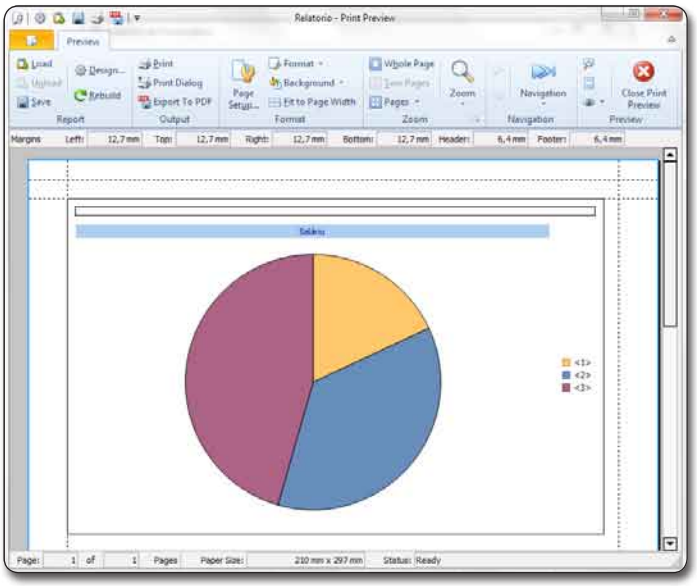


Figura 19. Impressão customizada por meio da Grid

Sander Carlos Mendes
sandermendes@gmail.com
Formação em Técnico em Informática pela Escola Técnica Estadual Dr. Júlio Cardoso. Trabalha como desenvolvedor independente. Cria aplicações em Delphi desde 2006, baseadas na arquitetura cliente/servidor com diversos tipos de banco de dados.

Site oficial da suíte de componentes
http://www.devexpress.com/

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Metropolis UI Applications

Desenvolvendo aplicações Delphi compatíveis com o Windows 8

A inserção do Sistema Operacional Windows 8 no mercado trouxe consigo uma nova onda de aplicativos, projetados para trabalhar tanto na forma tradicional, com o uso de mouse, quanto em telas sensíveis ao toque e habilitados a operar em ambientes Desktop e Mobile. A fim de acompanhar esta tendência, a Embarcadero apresentou recentemente a nova versão do Delphi, denominada XE3 e que trouxe, dentre suas novidades, um novo recurso chamado Metropolis UI (**Nota do DevMan 1**), que habilita as aplicações Delphi tradicionais, VCL e FireMonkey (FMX), a estarem compatíveis com o Windows 8. Sendo assim, neste primeiro momento, a comunidade Delphi, através do XE3, já está apta a produzir aplicativos Desktop compatíveis com o novo Sistema Operacional. Num futuro próximo, o esperado é que esse suporte se expanda, possibilitando a produção de aplicações Delphi para Mobile e WinRT (**Nota do DevMan 2**).

Nota do DevMan 1

Metro UI, ou simplesmente Metro, foi o codinome dado à nova interface de usuário desenvolvida pela Microsoft para o Sistema Operacional Windows 8. Dentre suas características, ela remete a termos como simplicidade, clareza e modernidade. Este tipo de interface foi primeiramente vista na plataforma móvel Windows Phone 7, através do uso de uma linguagem de design Metro, toda baseada em tipografia. Em agosto de 2012 o termo "Metro" foi extinto pela Microsoft, sob a justificativa de que se tratou apenas de uma denominação inicial. Todavia, ainda é notório o vínculo existente, no meio popular, entre o referido termo e o Sistema Operacional.

Nota do DevMan 2

WinRT, ou Windows Runtime, é o novo Framework de programação da Microsoft relacionado às novas aplicações "Metro" do Windows 8. Ele não se trata de uma nova camada de abstração do kernel do Windows, mas sim um novo runtime que faz acesso direto aos serviços do núcleo do Sistema Operacional, da mesma forma que a tradicional API Win32. Portanto, uma aplicação WinRT é dita como "nativa" do Windows 8, uma vez que é a forma mais direta do Sistema Operacional expor seus serviços a uma aplicação instalada.

Resumo DevMan

Metropolis UI Applications:

Com o advento do Sistema Operacional Windows 8, um novo paradigma de usabilidade e interface de usuário foi apresentado ao grande público. Este novo "estilo" é tratado como Metro e dá início a uma nova tendência de aplicações para a plataforma, causando um grande impacto no cenário comercial e de oportunidades. Como reflexo, na medida em que o Delphi é reconhecidamente tido como um provedor de excelência de aplicativos Windows, sua mais recente versão (XE3) apresenta novos recursos a fim de acompanhar toda esta evolução. O ponto chave para isto vem sob o nome de Metropolis UI que, em síntese, possibilita a criação de aplicações Delphi tradicionais, adequadas aos padrões do "novo" Windows. Portanto, neste artigo será visto todo o processo de desenvolvimento, bem como as novidades relacionadas às aplicações Metropolis UI (Metropolis UI Applications) no Delphi XE3.

Em que situação o tema é útil:

É fato que o sistema operacional da Microsoft, o Windows, abrange boa parte do mercado comercial em geral e, como consequência disso, abre espaço para a atuação de aplicações Delphi. Esta ligação que vem desde o nascimento da ferramenta, se mantém fortemente ativa nos dias atuais, sendo o principal fator de produção de tais aplicativos. Agora com o recente lançamento do Windows 8, uma nova tendência de mercado é apresentada, e isso, por questões óbvias, também se reflete no Delphi. Para mostrar de que forma isso acontece, este artigo dá maiores detalhes sobre o novo recurso trazido pelo Delphi XE3, denominado Metropolis UI, que provê a construção de aplicações Delphi compatíveis com o novo padrão determinado pelo Windows 8.

Tipos de projetos Metropolis UI

O Delphi XE3 traz em seu repositório de objetos, dois novos modelos pré-definidos de projetos, exclusivos para a criação de aplicações Metropolis UI – FireMonkey Metropolis UI Application e VCL Metropolis UI Application – conforme mostrado na **Figura 1**. Contudo, vale frisar desde o início que tais opções não estão relacionadas a uma nova tecnologia, sendo apenas uma forma mais prática e intuitiva que o IDE provê para a construção de aplicações VCL ou FireMonkey tradicionais, já compatíveis

ao Windows 8. Isso quer dizer que todos os formulários, componentes e controles utilizados são os mesmos já habituais em aplicações tradicionais, tal como TLabel e TActionList (no caso da VCL), porém já estilizados e direcionados ao padrão do novo Sistema Operacional.

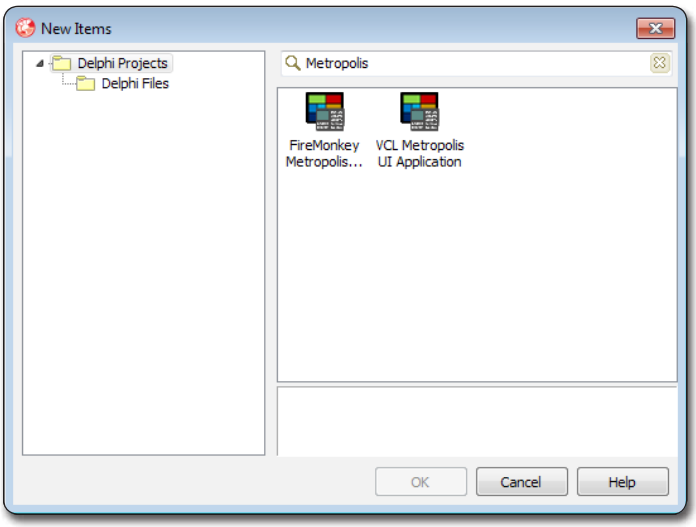


Figura 1. Novas opções de projeto – Metropolis UI Application

VCL Metropolis UI Applications

Esta opção cria um projeto VCL Forms tradicional, porém com recursos e estilização baseada no novo conceito de UI trazido pelo Windows 8. O próprio IDE se encarrega de montar um escopo padrão, facilitando a continuidade do desenvolvimento da aplicação. Olhando a lista de uses do formulário principal criado, comprova-se a ausência de qualquer namespace específico ou diferente dos habituais. De forma oposta, o que se vê são as já conhecidas Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls e Vcl.StdCtrls.

FireMonkey Metropolis UI Applications

Esta opção irá criar toda a estrutura inicial de uma aplicação FireMonkey, baseada no estilo Metropolis UI. Isso se reflete na estilização de alguns controles tradicionais (Ex: TLabel), na utilização de novos controles específicos (Ex: TMetropolisUIListBoxItem) bem como o acréscimo automático de outros componentes (Ex: TStyleBook). Os formulários são todos baseados em FMX.Forms.TForm.

Wizards Metropolis UI

Uma vez acionados, os novos modelos de projeto Metropolis UI proveem ao usuário um assistente (Wizard) que irá guiá-lo pela definição inicial do projeto. Essencialmente isso irá definir o layout da aplicação a ser construída, ou seja, o seu modelo de interface de usuário. Conforme pode ser visto na **Figura 2** são três as opções de "templates" disponibilizadas – Blank, Grid e Split Pane.

Conforme é possível supor, a opção Blank cria uma aplicação "em branco", sem a pré-definição de um determinado tipo de layout. Todavia, vale ressaltar que para todos os templates, incluindo o Blank, o Delphi se encarrega de habilitar o mínimo necessário para que a aplicação seja compatível ao Windows 8, incluindo assim o suporte a toque e estilização básica dos formulários. Já a opção Grid cria uma aplicação baseada em dois formulários, numa relação mestre-detalhe, apoiado num layout ao estilo grade (linhas e colunas) de rolagem horizontal (**Figura 3**). Em tempo de execução, no Form principal ficam dispostos vários grupos de itens, cada qual com sua imagem, título e subtítulo relacionado. Cada item possui então um conteúdo de detalhes atrelado, que é mostrado no formulário de detalhes criado junto da aplicação, e que é acessado ao se clicar sobre a imagem do item. Por fim, a opção Split Pane também se baseia em dois formulários e num layout de grade, mas agora numa relação de janela primária e secundária. Sendo assim, no Form principal (primário) ficam dispostos os grupos, representados por suas imagens, títulos e subtítulos associados. Em runtime, somente quando se clica sobre a imagem de cada um é que são mostrados seus itens e eventuais detalhes. Esta exposição se dá então no formulário secundário, que apresenta um layout dividido, conforme pode ser visto na **Figura 4**.

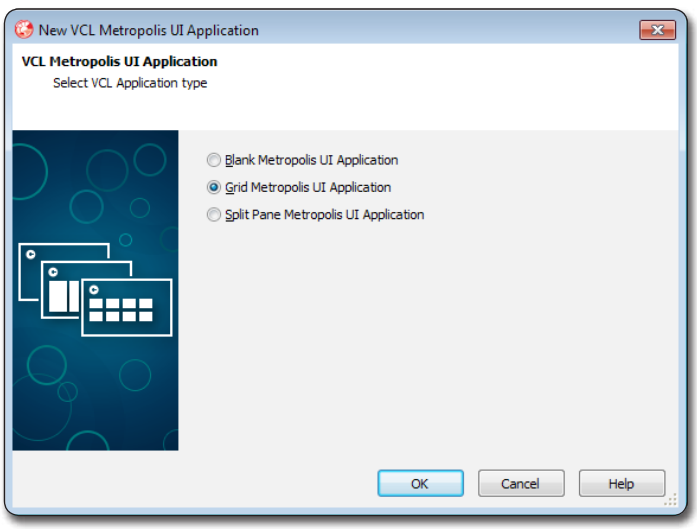


Figura 2. Opções disponíveis a um projeto VCL Metropolis UI Application

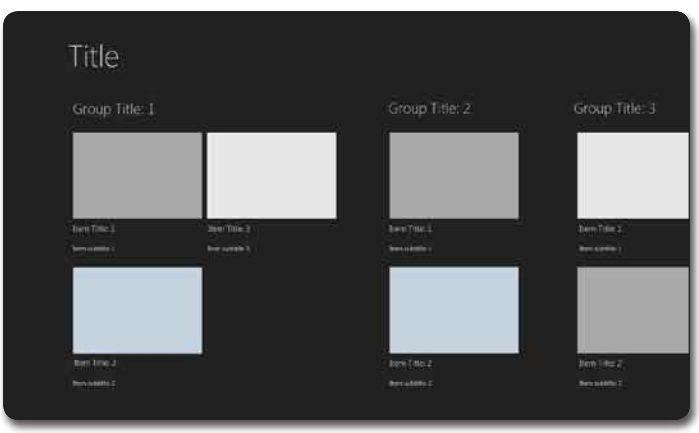


Figura 3. Grid Layout

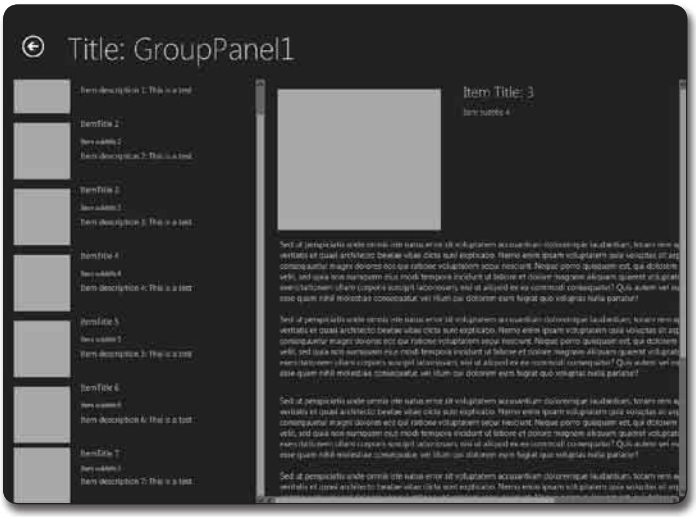


Figura 4. Split Pane Layout

Metropolis UI Forms

Pelo fato de uma aplicação Metropolis UI ser construída com base em um tipo de projeto tradicional VCL ou FireMonkey, a adição de um novo formulário ao projeto possibilita ao desenvolvedor duas opções. A primeira, e talvez a não recomendada, é a adição de um Form comum, tal como um VCL Form ou FireMonkey Form. Nestes casos, o formulário criado não possuirá o estilo Metropolis, quebrando parcialmente a compatibilidade da aplicação com o padrão do Windows 8. Sendo assim, como segunda opção surgem dois novos tipos de formulários, disponíveis no repositório de objetos do IDE, exclusivos ao estilo das aplicações Metropolis: VCL Metropolis UI Form e FireMonkey Metropolis UI Form. Conforme é possível supor, tais Forms nada mais são do que os formulários tradicionais, já estilizados e com suporte a toque, semelhantes aos criados juntos das aplicações VCL Metropolis UI e FireMonkey Metropolis UI, respectivamente. Adicionalmente, mesmo não sendo uma boa recomendação, cada um destes novos formulários pode também ser adicionado a um projeto VCL ou FireMonkey tradicional. Somente para fins de curiosidade, a seguir são mostradas algumas das configurações relativas a um VCL Metropolis UI Form, onde é possível notar, entre outras coisas, a ausência da borda (BorderStyle = bsNone), e seu estado inicial como Maximizado (WindowState = wsMaximized), além do vínculo com um componente TGestureManager (ver **Listagem 1**).

Listagem 1. Arquivo .dfm de um formulário com Metropolis UI

```
object Form1: TForm1
 ...
 BorderStyle = bsNone
 Font.Name = 'Segoe UI'
 Touch.GestureManager = GestureManager1
 Touch.InteractiveGestureOptions = [igoPanSingleFingerHorizontal, igoPanInertia,
  igoParentPassthrough]
 WindowState = wsMaximized
 ...
end
```

Controles VCL e FireMonkey para Metropolis UI

Mesmo o surgimento das Metropolis UI Applications, a biblioteca de componentes da VCL se manteve quase que intacta, não provendo nenhum tipo de controle específico a este novo formato. No entanto, visando manter um aspecto visual condizente com o estilo Metro, os controles VCL utilizados em uma aplicação Metropolis UI sofrem algumas estilizações, afetando principalmente seu tamanho e sua fonte. Numa exemplificação simples, citando o uso de um TEdit comum em uma aplicação Metropolis, este apresenta sua propriedade Height com o valor 33, ao contrário do valor 21 de uma aplicação VCL Forms padrão. Já no quesito Fonte, as aplicações Metropolis UI, sejam elas VCL ou FireMonkey, fazem uso da fonte padrão do Windows 8 denominada Segoe UI.

De forma oposta, o FireMonkey foi estendido, implementando algumas especificações de controles para Metropolis UI. Como exemplo, é possível citar o novo controle TMetropolisUIListBoxItem, que nada mais é do que um item de ListBox, estilizado para Metro, conforme mostra a **Figura 5**. Na mesma medida, alguns controles existentes apresentam agora algumas customizações adicionais específicas, centralizadas numa propriedade de nome StyleLookup, comum a controles como TEdit, TLabel e TButton. Nela fica disponível uma lista de "estilos" que alteram o visual padrão do controle de acordo com a funcionalidade escolhida. Para o TEdit, por exemplo, algumas das opções disponíveis são:

- ClearingEditStyle: edit com botão interno para apagar o texto contido no controle;
- EditStyle: edit padrão;
- ComboEditStyle: edit com estilo de ComboBox;

Figura 5. ListBox contendo itens do tipo TMetropolisUIListBoxItem

Conceitos Metropolis UI

Como toda grande novidade que surge no meio tecnológico, o novo paradigma de interfaces de usuário introduzido pelo Windows 8, associa as aplicações deste tipo uma série de novos conceitos relacionados. Sendo assim, a construção plena de aplicações Metropolis UI no Delphi exige a assimilação de alguns deles, conforme será visto a seguir.

Application Bar

Uma das características das aplicações Metro é a ausência de uma barra de menu, comumente localizada na parte superior

da janela de aplicativos Windows tradicionais. Em substituição, é utilizado um novo controle chamado de Application Bar, ou simplesmente AppBar. Por padrão, esta barra não fica visível na interface de usuário do aplicativo num primeiro momento, vindo à tona mediante a comandos de tecla, gestos ou mouse. Dessa forma, a partir de um formulário básico de uma aplicação Metropolis UI construída no Delphi, é possível se acionar a AppBar, por exemplo, pressionando-se a tecla ESC. Esta ação faz com que apareça no rodapé da janela uma barra contendo um botão X, conforme mostra a **Figura 6**, que serve para encerrar a aplicação.

Figura 6. AppBar de uma aplicação VCL Metropolis UI

No Delphi, em uma aplicação VCL Metropolis UI, a implementação da AppBar se dá pelo uso de um controle já conhecido do desenvolvimento tradicional, o TPanel. Sendo assim, ao se utilizar as opções de Grid ou Split Pane providos pelo Wizard Metropolis UI provido pelo IDE, o formulário principal criado já traz um TPanel na forma de AppBar, tal como mostra o código a seguir, referente ao trecho de declaração da classe do Form:

```
TSplitForm = class(TForm)
...
  AppBar: TPanel;
```

De forma semelhante, a AppBar numa aplicação FireMonkey Metropolis UI é implementada através da combinação do uso dos controles TToolBar, TPopup e TLayout.

Live Tiles

Em meio ao novo paradigma de interfaces de usuário proporcionado pelo Windows 8 está também o conceito de Live Tiles. De forma rápida, podemos defini-los como sendo grandes e coloridos blocos informativos que se fazem presentes na tela inicial do novo Sistema Operacional (**Figura 7**). Funcionalmente, um Live Tile pode ser associado a uma aplicação, de forma a servir como um link para ela, ou mesmo para exibir suas informações mais recentes de status e atualização. Em suma, ele acaba por se tornar uma representação do aplicativo na Start Screen do Windows 8. Vale ressaltar que os Live Tiles, diferentemente das aplicações Metropolis UI, são aplicações WinRT, sendo consideradas nativas do Sistema Operacional.

Sendo assim, visando atender a todos estes requisitos, o Delphi XE3 traz consigo novos recursos que proporcionam ao desenvolvedor uma maneira fácil de construir e trabalhar com Live Tiles em aplicações Desktop Metropolis UI (VCL ou FireMonkey). Na prática, todo este processo se concentra sobre o novo componente TLiveTile, o único presente no grupo Metropolis UI da Tool Palette (**Figura 8**), através de uma série de camadas de serviços.

Figura 7. Live Tiles no Windows 8

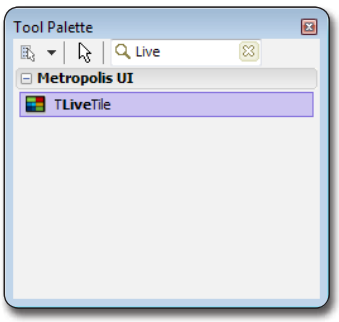


Figura 8. Componente TLiveTile

Internamente, a aplicação Delphi faz uso do componente TLiveTile para se associar e, consequentemente, poder ter acesso a um Live Tile. A seguir, o componente direciona as informações de atualização da aplicação a um serviço denominado MetropolisSvc, que nada mais é do que um serviço REST local, responsável pela comunicação entre o aplicativo e o LiveTile. O detalhe aqui é que para se fazer jus ao conceito "Live" (vivo) relacionado aos Live Tiles, o serviço realiza o envio das informações a cada minuto, num processo totalmente contínuo.

De forma complementar, quando o usuário final clica sobre o LiveTile relacionado, presente na interface do Windows 8, uma mensagem é enviada ao serviço MetropolisSvc. Por conseguinte, este notifica uma aplicação complementar, denominada LaunchAgentHost, que fica com a incumbência de que a aplicação Metropolis UI seja exibida em primeiro plano na área de trabalho do Windows.

Tutorial

Criando um Live Tile no Delphi XE3

As etapas que envolvem a criação de um Live Tile no Delphi XE3, apesar de simples, apresentam alguns detalhes que irão determinar o sucesso da operação. Pelo fato de ser um recurso totalmente novo, torna-se importante para o aprendizado que cada passo do processo seja devidamente exposto e comentado. Dito isto, a primeira etapa da criação de um novo Live Tile é o envolvimento de um projeto VCL Metropolis UI Application ou FireMonkey Metropolis UI Application, seja ele novo ou existente. A seguir, conforme já mencionado anteriormente, o responsável por todo o trabalho com Live Tiles no Delphi é o componente TLiveTile que, por questões óbvias, deverá então estar presente na aplicação.

De posse do componente, o preenchimento de duas de suas propriedades – CertificateName e PublisherName – se torna obrigatório, uma vez que ambas fazem parte de um controle interno do processo, seguindo uma política estabelecida pela fabricante do Sistema Operacional (Microsoft). Até mesmo por este fato, o próprio Delphi já determina um valor padrão fictício para elas, conforme mostrado a seguir (**Listagem 2**). A fim de exemplificação, isso já é o suficiente.

Listagem 2. Configuração do LiveTile

```
object LiveTile1: TLiveTile
  ...
    PublisherName = 'tileproxy.test'
    CertificateName = 'tileproxy.test'
  ...
end
```

Além dessas, há ainda outras duas propriedades que são de extrema importância, no que diz respeito ao aspecto visual do Live Tile que está sendo criado – Narrow Tile e Wide Tile. Estes são os nomes dos dois tamanhos distintos que o Windows 8 trabalha por padrão, e que varia de acordo com o gosto do usuário final, já que estes podem redimensionar cada Live Tile em tempo de execução. Pelo Delphi, é possível ainda restringir a aplicação a trabalhar somente com o tamanho Narrow, bastando para isso apenas marcar sua outra propriedade IncludeWideTile como False. Para cada tamanho, é possível se configurar um template diferente, através da sub-propriedade TileTemplate, que provê uma lista de modelos já pré-definidos. Por exemplo, para Wide Tile, algumas das opções disponíveis podem ser vistas na **Figura 9**.

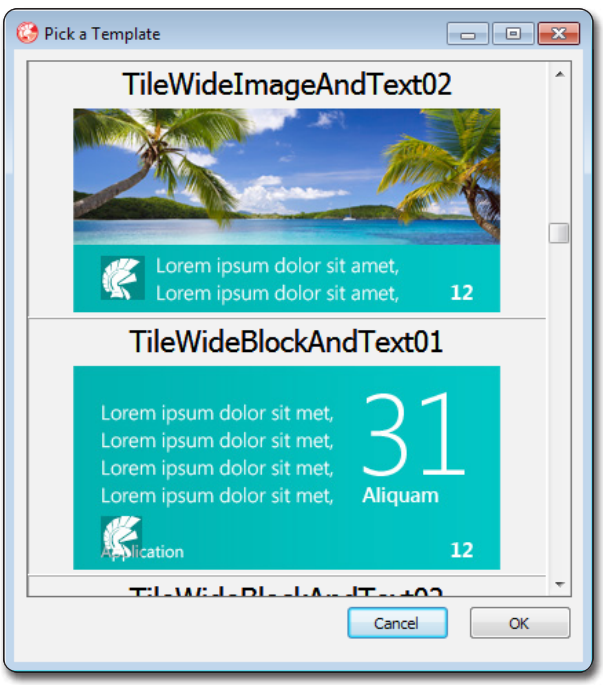


Figura 9. Opções de Tile Templates

Uma vez configurado o componente, o próximo passo é criar todo o "código-fonte" do Live Tile. Este processo é feito através de uma opção pré-estabelecida, denominada Instrument Project, localizada no menu de contexto do próprio componente. Ao acionar este recurso, os arquivos relacionados ao Live Tile são todos criados de forma automática pelo IDE, sendo gerados em uma pasta localizada no mesmo diretório do projeto Metropolis UI, intuitivamente nomeada como TileSource. Analisando de forma superficial os arquivos criados (**Figura 10**), é possível perceber que eles não estão relacionados a um projeto Delphi, uma vez que os Live Tiles são aplicações baseadas no novo Windows Runtime. Portanto, o que o IDE faz é apenas gerar o mínimo necessário para que se produza um Live Tile associado à aplicação Delphi (Metropolis UI) que está se construindo.

Dentre os arquivos gerados, um merece destaque especial em razão da função que exerce no contexto geral. Trata-se do arquivo

AppxManifest.xml, que é um arquivo XML de marcações simples, que determinam os atributos do Live Tile. Na **Listagem 3** é mostrado um trecho de seu código, onde é possível ver algumas das propriedades especificadas ao Live Tile. O primeiro valor, marcado como <DisplayName>, determina o nome de visualização do projeto, sendo o texto que será exibido no Live Tile já no ambiente do Windows 8. Na mesma medida, o valor <Logo> indica a imagem que será utilizada na exibição, junto do texto. Perceba que ela aponta para a pasta Assets, que contém as imagens utilizadas no Live Tile. Por fim, a propriedade <PublisherDisplayName> reflete o valor da propriedade PublisherName do componente TLiveTile, mencionada anteriormente.

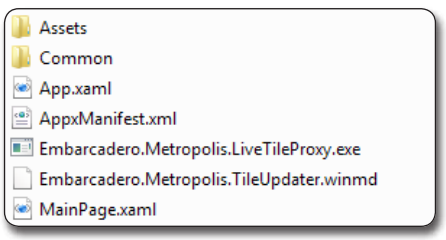


Figura 10. Arquivos gerados na pasta TileSource

Listagem 3. Trecho de código presente no arquivo AppxManifest.xml

```
<Properties>
      <DisplayName>Project1Metropolis</DisplayName>
       <PublisherDisplayName>tileproxy.test</PublisherDisplayName>
       <Logo>Assets\StoreLogo.png</Logo>
</Properties>
```

Apesar de se ter acesso ao seu conteúdo, é importante ter em mente que o arquivo AppxManifest.xml é atualizado de forma automática pelo IDE, baseado nas propriedades do componente TLiveTile. Sendo assim, é uma boa prática voltar as atenções somente a ele, e não diretamente sobre o arquivo propriamente dito. Portanto, para se alterar o nome de exibição do projeto, por exemplo, basta realizar a alteração na propriedade MetropolisAppName do componente. Uma vez feito isso, para que essa alteração seja refletida no arquivo de atributos (.xml), é necessário acionar o item Update App Manifest do menu de contexto do próprio componente.

O passo final do processo de criação do Live Tile da aplicação Metropolis UI é efetivamente criá-lo e, logo, instalá-lo na Start Screen do Windows 8. Tudo isso é feito também de forma automática pelo Delphi, através da opção Deploy Tile do componente TLiveTile.

Vale ressaltar que esta opção de desenvolvimento de Live Tiles para as aplicações Metropolis UI somente é habilitada por completo, mediante a configuração adequada do ambiente de desenvolvimento. De prima, tem-se a obrigatoriedade da presença do Windows 8 como Sistema Operacional. A partir disso, algumas das opções de desenvolvimento e deployment, tal como a citada Deploy Tile, só estarão disponíveis nestes cenários. Por questões de espaço, o processo de configuração do ambiente está fora do escopo deste artigo. Além disso, por se tratar de um processo não tão trivial, envolvendo vários passos a serem meticulosamente seguidos, a Embarcadero disponibiliza em meio a documentação online do Delphi uma página exclusiva tratando do assunto (ver link ao final do artigo).

Convertendo uma aplicação tradicional VCL Forms

Anteriormente ao surgimento dos recursos recentemente incorporados ao Delphi, tais como FireMonkey e Metropolis UI, a grande base de aplicativos produzidos pela comunidade se traduzia em aplicações VCL para Win32. Sendo assim, visando acompanhar as oportunidades de mercado mais atuais, muitas destas aplicações necessitaram ser convertidas para FireMonkey, ainda na época de seu lançamento no Delphi XE2. Em função disso, nesta mesma época, surgiram na rede vários conversores de projetos VCL para FireMonkey, inclusive um distribuído pela própria Embarcadero. De forma parecida, no cenário atual a comunidade Delphi vê nas Metropolis UI Applications uma nova possibilidade de mercado, em razão do lançamento do Windows 8. Conforme visto anteriormente, uma aplicação Metropolis UI nada mais é do que uma aplicação tradicional (VCL ou FMX) "estilizada" ao novo padrão de UI introduzido pelo Windows 8. Todavia, esta estilização só acontece de forma automática através do uso dos Wizards Metropolis UI. Em outras palavras, somente para projetos novos. A "conversão" de um projeto VCL Forms ou FireMonkey tradicional existente exige alguns ajustes manuais para que a aplicação produzida seja considerada uma aplicação Metropolis UI. Mediante esta situação, e visando atender o grande público, será mostrado agora um exemplo prático, em três passos, do processo de "conversão" de uma aplicação VCL Forms existente, para Metropolis UI.

1º Passo

De posse de uma aplicação VCL Forms tradicional, o primeiro passo do processo é a conversão de seus formulários para o estilo Metropolis UI. Já prevendo esta necessidade, o próprio Delphi disponibiliza uma opção pronta para tal, denominada Convert to Metropolis UI (**Figura 11**), disponível no menu de contexto do próprio Form. Feito isso, o formulário atual é adequado ao padrão de UI do Windows 8, através da simples customização de algumas de suas propriedades, tal como BorderStyle = bsNone, que define o formulário sem bordas, e WindowsState = wsMaximized, para que seja sempre iniciado maximizado. O detalhe aqui é que esse processo envolve apenas um único Form da aplicação e, portanto, deverá ser repetido a cada formulário contido no projeto.

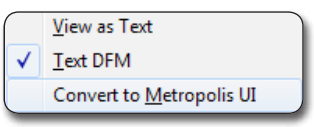


Figura 11. Opção Convert to Metropolis UI

No instante seguinte é aberta a janela de opções de projeto, onde deve-se definir uma aparência Metro à aplicação. Isso é feito pela seleção de um, dentre os quatro estilos Metropolis UI disponíveis (**Figura 12**).

Por fim, uma boa prática é adicionar a seguinte instrução no código do arquivo .dpr do projeto (**Listagem 4**), que faz com que seja mostrado um teclado em tela, para os componentes voltados à entrada de dados do usuário, nas situações em que se fizer necessário.

2º Passo

Voltando aos formulários da aplicação, o segundo passo da conversão é a criação das AppBars, que substituem as tradicionais barras de menu da aplicação. Conforme vimos, num projeto VCL Metropolis UI ela é implementada através da utilização de um controle TPanel, e é justamente isso que deverá ser feito. Portanto, ao incluir um novo TPanel no formulário, defina suas propriedades conforme a seguir (**Listagem 5**). A intenção aqui foi a de utilizar as configurações semelhantes às propostas pelo Wizard de criação das VCL Metropolis UI Applications.

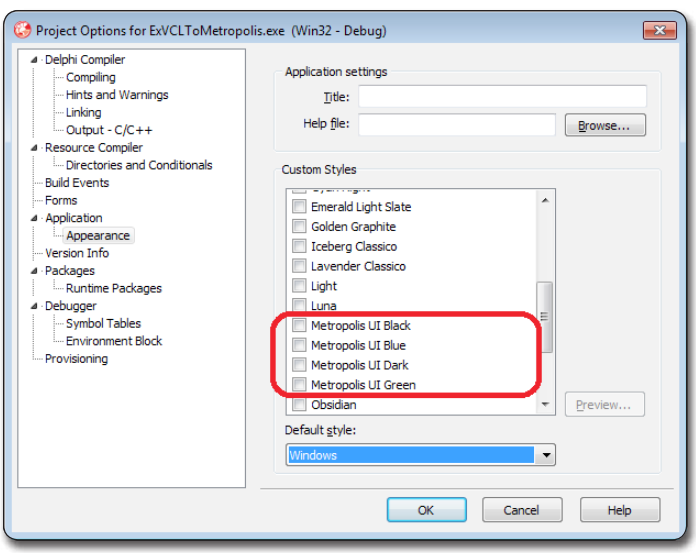


Figura 12. Metropolis UI Styles disponíveis

Listagem 4. Configuração do arquivo .dp

```
begin
  Application.Initialize;
  Application.UseMetropolisUI;
end;
```

Listagem 5. Definição de um controle TPanel

```
object AppBar: TPanel
  ...
  Width = 434
  Height = 75
  Align = alBottom
  AutoSize = True
  Caption = ''
  Visible = False
end
```

O último ajuste com relação à AppBar fica a cargo da utilização de botões. Nas VCL Metropolis UI Applications, tais botões nada mais são do que imagens, providas por um componente TImage. A quantidade e o tipo de botão a ser utilizado ficam a critério do desenvolvedor. A fim de contribuir com o desenvolvimento, o Delphi traz consigo alguns ícones para serem utilizados em projetos Metropolis UI, tal como em botões. Por padrão, esta biblioteca fica localizada em:

C:\Users\Public\Documents\RAD Studio\10.0\Styles\VCLMetropolisUIIconLibrary

3º Passo

Até aqui a aplicação VCL Forms tradicional que está sendo convertida já possui toda a aparência de uma aplicação Metropolis UI. No entanto, conforme mencionado anteriormente, outra característica fundamental de uma aplicação Metro é seu recurso de toques e gestos, a fim de atender cenários sensíveis ao toque, tal como em Tablets. No Delphi, este recurso é provido pelo componente TGestureManager, que deverá então ser "ligado" ao formulário, através da propriedade Touch.GestureManager do mesmo. Paralelamente, se faz também necessária a presença de um componente TActionList, que será usado para mapear em ações os gestos que serão definidos. Um exemplo disso pode ser visto na **Figura 13**, onde foram definidos dois gestos padrões (Up e Down), que foram associados a uma ação criada (actAppBar) no ActionList. Isso significa que ao fazer os movimentos de tais gestos, aquela determinada ação será executada.

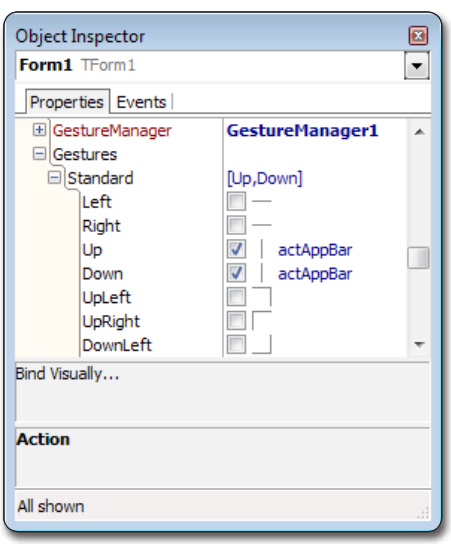


Figura 13. Associando gestos a ações

Este é o princípio utilizado na definição do comportamento de mostrar/ocultar a AppBar em aplicações Metropolis UI. Isto porque, por padrão, ela deve começar oculta em tela, sendo exibida somente a partir de ações feitas pelo usuário final. Portanto, aproveitando a ação citada anteriormente, em seu evento OnExecute, podemos definir tal comportamento da AppBar, conforme mostra a **Listagem 6**.

De forma básica, isso é o suficiente para que tenhamos uma aplicação Metropolis UI, sem fazer uso do Wizard disponibilizado pelo Delphi. Pela **Figura 14** é possível ver o resultado final da conversão realizada nos três passos mostrados.

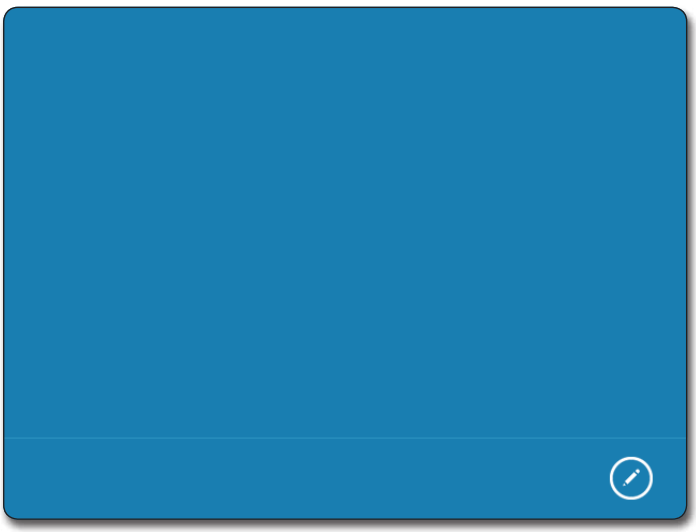


Figura 14. VCL Metropolis UI Application

Listagem 6. Código do evento OnExecute da ação actAppBar

```
procedure TForm1.actAppBarExecute(Sender: TObject);
begin
  AppBar.Visible := not AppBar.Visible;
end;
```

Conclusão

Este artigo tentou desmitificar uma das grandes novidades trazidas pela versão XE3 do Delphi que é o recurso de Metropolis UI. Conforme visto, novamente o IDE e a tecnologia conseguem manter as coisas simplificadas, mas não menos eficientes, provendo os recursos necessários para o desenvolvedor se manter em dia com as atuais tendências de mercado. É de conhecimento de todos que o foco do momento está voltado ao Windows 8, a mais nova versão do Sistema Operacional da Microsoft, e que está ditando uma nova geração de aplicativos. Mediante isso, a comunidade Delphi não fica atrás, conseguindo produzir aplicações compatíveis com este novo "estilo", utilizando os mesmos tipos de projetos habituais (VCL ou FMX), apoiados pelo novo recurso de Metropolis UI. Por esta razão, a curva de aprendizado é muito reduzida.

O desafio está lançado, com a certeza de que no momento da escrita deste texto, muitas aplicações Delphi já foram e continuam sendo produzidas, direcionadas para esta nova plataforma do Windows 8. Sendo assim, mais um mercado de abrangência é atingido pelo Delphi tornando-se, consequentemente, mais uma vertente comercial a ser explorada.

Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 7 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas. Certified Delphi Developer. Certified Embarcadero Instructor. Blog: prismsilver.blogspot.com

Delphi XE3 – Download Trial
https://downloads.embarcadero.com/free/delphi

Delphi XE3 – Página oficial do produto
http://www.embarcadero.com/products/delphi

Configurando o ambiente para o desenvolvimento Live Tiles para Metropolis UI
http://docwiki.embarcadero.com/RADStudio/XE3/en/Setting_Up_Your_Environment_for_Developing_Live_Tiles_for_Metropolis_UI#Steps_to_Set_Up_Development_Environment

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

DORM

Empregue ORM nos projetos com o auxílio deste Framework

Resumo DevMan

DORM:

Contar com ferramentas que auxiliem ao máximo o processo de desenvolvimento de um Software sempre é um fator diferencial. O Framework DORM é uma grande alternativa quando existe a necessidade de empregar ORM através do Delphi, uma vez que é Open Source e possui uma estrutura sólida, trazendo diversos benefícios, além de contar com vários colaboradores que se empenham em adicionar novas funcionalidades ao mesmo, ou mesmo efetuar possíveis correções. Neste artigo, serão apresentados os conceitos básicos para empregar o DORM nos projetos, demonstrando como inserir, recuperar, consultar e apagar informações utilizando-o.

Em que situação o tema é útil:

Auxiliar no desenvolvimento de soluções OO, implementando o recurso de ORM, aumentando assim a produtividade e tornando a aplicação mais flexível, delegando ao DORM a camada de conexão com o banco de dados e o mapeamento das entidades.

A Orientação a Objetos, um conjunto de diversas metodologias e tecnologias que permitem o desenvolvimento de aplicações otimizadas e melhor organizadas, cada vez mais ganha adeptos no ambiente Delphi. Com o passar do tempo, o RAD recebeu uma diversidade de novos recursos e Frameworks que possibilitaram o desenvolvimento de aplicações OO de forma semelhante a que ocorre em outros ambientes de desenvolvimento. Funcionalidades como RTTI (Run-Time Type Information), diversas informações mantidas na memória sobre os tipos de dados dos objetos, LiveBindings, ligações entre os objetos com base em expressões, Generics, tipos parametrizados entre outros. Todos estes também se tornam fortes aliados para auxiliar o emprego de OO através do Delphi. Sendo assim, é possível mapear as entidades que fazem parte do mundo real para o ambiente computacional, onde as regras de negócio podem ser "traduzidas" para as classes. Diante deste cenário, objetos podem ser instanciados destas classes para possibilitar toda a manipulação das informações fornecidas pelo usuário através do Software. Contudo, tais objetos ficam residentes na memória volátil, tal como a RAM, e evidentemente, necessitam de técnicas e mecanismos que possibilitem a gravação dos mesmos para seus respectivos lugares em uma memória definitiva, como é o caso do disco rígido, permitindo a futura recuperação da informação. O recurso básico que possibilita este processo é denominado persistência, possibilitando a existência dos objetos além do tempo de execução do Software. Normalmente, as informações persistidas são armazenadas em um banco de dados relacional, mas outros repositórios também podem ser empregados. Existem diversas técnicas de persistência que podem ser empregadas, tal como o DAO (Data Access Object), onde classes são utilizadas para encapsular e abstrair toda a estrutura de acesso ao banco de dados, fornecendo interfaces simplificadas para a realização das operações (CRUD) dos objetos. Entretanto, um recurso mais interessante que pode ser amplamente empregado quando o assunto é persistência de dados, é o Mapeamento Objeto Relacional (ORM - Object Relational Mapping). Este conceito permite que objetos presentes no Software possam ser persistidos de uma maneira automática para as tabelas que constituem um banco de dados relacional. Diante deste cenário, o ORM se torna um grande aliado. Ele traz diversas vantagens, como é o caso do aumento da produtividade, melhor manutenibilidade, fácil migração do projeto para outro sistema de gerenciamento de banco de dados SGBD, dentre outros.

Um recente Framework Open Source que está em constante desenvolvimento e já possui diversas funcionalidades para o emprego de ORM através do Delphi é o DORM (Delphi ORM), criado por Daniele Teti. Atualmente, ele suporta a conexão com diversos sistemas de gerenciamento de banco de dados, como é o caso do Firebird, SQLite 3 e do SQL Server, empregando alguns Frameworks presentes no Delphi, como é o caso do dbExpress. Surge então uma nova camada na aplicação que separa os objetos do banco de dados, sendo responsável pela tradução dos mesmos, permitindo o emprego de persistência de uma forma automatizada. Apesar de ser um projeto recente, o DORM foi muito testado e possui alta performance.

De início, um ponto interessante ao empregar DORM nos projetos é com relação à modelagem, onde não há nenhuma interferência por parte do Framework. As classes não precisam herdar outras classes ou interfaces, nem existe a necessidade que

os atributos sejam definidos através de published, sendo assim, uma classe que será empregada neste Framework não tem sua estrutura alterada com relação à tradicional forma de definir as mesmas através do Delphi.

Com o DORM também é possível utilizar uma determinada base de dados para o desenvolvimento, outra para testes e outra para produção, onde as mesmas são especificadas no momento da criação de uma sessão, que por sua vez é a responsável pelo gerenciamento da conexão e das operações.

A definição da conexão com o banco de dados, do mapeamento, dos Logs, etc, é realizada através de arquivos baseados no padrão JSON (JavaScript Object Notation). Este padrão, originado da linguagem JavaScript que também é adotado em outras linguagens e é baseado em texto, é extensamente empregado para a transferência de dados e a definição de objetos de uma forma leve, prática e fácil de ser compreendida, através de pares compostos por nome-valor. Sendo assim, o desenvolvedor deve incluir em seu projeto, arquivos JSON contendo as informações necessárias para que o Framework possa ser empregado.

Entretanto, um dos pontos fracos do DORM é com relação à documentação, que é um pouco escassa, mas este não é um fator preocupante, pois, o Framework é fácil de ser compreendido e empregado, além de existirem diversos exemplos que acompanham os fontes e podem ser utilizados para seu entendimento.

Contudo, vale à pena frisar também que o Delphi conta com outros excelentes Frameworks para o emprego de ORM, alguns gratuitos e outros proprietários, tais como TMS Aurelius, tiOPF, hcOPF, InstantObjects, DObjects e etc.

Tutorial

O SQLite

Antes de iniciar o projeto com ORM através do Delphi, primeiramente será criado um banco de dados SQLite. O SQLite é uma biblioteca de Software autossuficiente e multiplataforma que é baseada em C e implementa um banco de dados SQL que não necessita ser instalado ou de um servidor. Seu código fonte é disponibilizado livremente. Quanto a suas funcionalidades, o SQLite possui diversos recursos como transações isoladas e em versões mais recentes, chaves estrangeiras, além de suportar bases de dados com Terabytes de informações. Atualmente, é muito popular no ambiente de desenvolvimento móvel, pois, está presente de forma nativa no sistema operacional Android, mas pode ser utilizado para as mais variadas finalidades.

Quanto a criação e manutenção de um banco de dados SQLite, existem diversas ferramentas gratuitas e pagas que podem ser empregadas. O Software utilizado neste projeto para a elaboração do banco de dados e sua estrutura foi o SQLite Administrator, que é gratuito e pode ser obtido facilmente (ver seção **Links**), além de ser em Português. A **Listagem 1** apresenta o código SQL empregado para a definição de uma simples tabela denominada Pessoa, para o exemplo que será apresentado.

Como é possível notar, a tabela Pessoa foi definida de uma forma simples e não foi utilizado o recurso de autoincremento para a chave-primária, embora o SQLite3 ofereça tal funcionalidade nativamente, a sequência da chave será realizada através dos recursos do Framework DORM.

Listagem 1. Código SQL para criação das tabelas

```
CREATE TABLE PESSOA (
  ID       INTEGER NOT NULL PRIMARY KEY,
  NOME     VARCHAR(50),
  ENDERECO VARCHAR(50)
);
```

DORM

Para começar a utilizar o DORM, basta apenas efetuar o download de seus fontes, disponíveis no Google Code (ver seção **Links**), e posteriormente incluir alguns diretórios no Library Path do Delphi, tais como "trunk\source" e "trunk\library", não necessitando de instalação ou configurações adicionais. Sendo assim, para dar início a um exemplo prático com o DORM, uma nova aplicação FireMonkey deve ser criada através do menu File>New>FireMonkey HD Application, lembrando que este Framework pode ser empregado em ambas as plataformas, FMX e VCL. O primeiro passo para o emprego do DORM em um projeto é a elaboração das classes necessárias, um processo que pode ser efetuado da mesma forma que é realizado em outras ocasiões conforme mencionado anteriormente, empregando PODO (Plain Old Delphi Object), por exemplo. Sendo assim, uma nova Unit foi adicionada ao projeto para armazenar a classe TPessoa e recebeu a denominação de UClasses. A **Listagem 2** apresenta o código fonte que foi utilizado para a definição da classe TPessoa.

Listagem 2. Código da classe TPessoa

```
type
  TPessoa = class
  private
    FId: Integer;
    FNome: string;
    FEndereco: string;
  public
    constructor Create(Nome, Endereco: string);
  published
    property Id: Integer;
    property Nome: string;
    property Endereco: string;
  end;
```

Como pode ser observado, a estrutura da classe foi definida de acordo com a tabela Pessoa criada anteriormente. Após a elaboração da mesma, basta pressionar simultaneamente as teclas CTRL + SHIFT + C para que o Delphi gere automaticamente os métodos set (write) para as propriedades da mesma, definindo

também o read, para possibilitar a leitura dos atributos privados. Também é possível notar que foi definido um método construtor para a classe com dois parâmetros cujo intuito é carregar o nome e o endereço da nova pessoa diretamente no momento da criação da mesma. Sendo assim, tal método deve conter o simples trecho de código a seguir:

```
inherited Create;
SetNome(Nome);
SetEndereco(Endereco);
```

Criada a classe, seus atributos e métodos, o próximo passo é elaborar os arquivos baseados em JSON para empregar os recursos do Framework DORM. Neste processo, qualquer editor de textos pode ser empregado, como é o caso do Bloco de Notas ou do Notepad++. O primeiro arquivo criado foi denominado dormsqlite3.conf, e será utilizado para armazenar as configurações básicas para o Framework. A **Listagem 3** apresenta a estrutura utilizada para o arquivo dormsqlite3.conf.

Listagem 3. Estrutura do arquivo JSON dormsqlite3.conf

```
01 {
02  "persistence": {
03    "development": {
04      "database_adapter": "dorm.adapter.Sqlite3.TSqlite3PersistStrategy",
05      "database_connection_string":"Base\\Base.s3db",
06      "keys_generator":"dorm.adapter.sqlite3.TSqlite3TableSequence",

07      "key_type":"integer", // "string" | "integer"
08      "null_key_value": 0},
09
10    "test": {
11      "database_adapter": "dorm.adapter.Sqlite3.TSqlite3PersistStrategy",
12      "database_connection_string":"Base\\Base.s3db",
13      "keys_generator":"dorm.adapter.sqlite3.TSqlite3TableSequence",
14
15      "key_type":"integer", // "string" | "integer"
16      "null_key_value": 0},
17
18    "release": {
19      "database_adapter": "dorm.adapter.Sqlite3.TSqlite3PersistStrategy",
20      "database_connection_string":"Base\\Base.s3db",
21      "keys_generator":"dorm.adapter.sqlite3.TSqlite3TableSequence",
22
23      "key_type":"integer", // "string" | "integer"
24      "null_key_value": 0},
25
26   "persistent_classes": ["TPessoa"]
27  },
28  "config": {
29    "logger_class_name": "dorm.loggers.FileLog.TdormFileLog"
30  }
31 }
```

Nesse arquivo de configuração podemos determinar como será a conexão com um banco de dados em três situações distintas do projeto: desenvolvimento, teste e produção. Assim, é possível utilizar um banco de dados diferente para cada uma dessas situações. Além da conexão também configuramos detalhes relativos à geração das chaves primárias. Em persistence (linha 2) são definidas e agrupadas essas configurações, temos o grupo development (linha 3), test (linha 10) e release (linha 18), em cada um desses grupos configuramos um acesso a banco de dados utilizando atributos específicos. Por exemplo, em database_adapter (linha 4) é atribuído o recurso que será responsável pela conexão com o banco de dados SQLite através do DORM. Em database_connection_string (linha 5), é definido o caminho para o banco de dados, que neste caso é o Base.s3db, presente no diretório Base, contido na mesma pasta do executável da aplicação. Como não foi empregado auto incremento nativamente para a chave primária da tabela Pessoa, em keys_generator (linha 6) foi especificado o recurso responsável pela geração das mesmas para o SQLite3, TSQLite3TableSequence. Na configuração key_type (linha 7), foi atribuído o tipo para a chave, Integer, e o valor 0 para null_key_value (linha 8). Posteriormente, através de persistent_classes (linha 26), é definida a classe (ou classes) que deve ser persistida pelo Framework (TPessoa). Para concluir, em config é definido o recurso que deve ser empregado para a geração automática dos arquivos de Log do DORM, que neste caso recebeu o valor de dorm.loggers.FileLog.TdormFileLog para que os Logs sejam gerados em um arquivo de texto no próprio diretório em que se encontra a aplicação. Vale a pena ressaltar que o DORM também permite a geração de Logs para a ferramenta CodeSite, dentre outros, lembrando que Log é um recurso que possibilita registrar e acompanhar as ações que são realizadas, facilitando o processo de identificação de falhas.

Terminada a definição das configurações básicas, deve ser criado um novo arquivo JSON para efetuar o mapeamento da classe. Neste projeto, ele recebeu a denominação de mapeamento.conf. A **Listagem 4** apresenta a estrutura do arquivo mapeamento.conf.

Listagem 4. Estrutura do arquivo JSON mapeamento.conf

```
{
  "TPessoa": {
    "package":"UClasses",
    "table": "PESSOA",
    "id": {"name":"Id", "field":"ID", "field_type":"integer"},
    "fields":[
      {"name":"Nome", "field":"NOME", "field_type":"string", "size": 50},
      {"name":"Endereco", "field":"ENDERECO", "field_type":"string", "size": 50}
      ]
  }
}
```

É possível notar que a classe TPessoa é definida, onde posteriormente, através de "package", é atribuído o nome da Unit que contém a classe criada anteriormente, UClasses. Em "table", é definida a tabela na qual os objetos desta classe deverão ser persistidos, neste caso Pessoa. Em seguida, "id" é utilizado para as definições da chave-primária da tabela em questão, onde é indicada a propriedade da classe atual que deve ser utilizada, o nome do campo na tabela e seu tipo. Para concluir, "fields" é um array que contém o mapeamento para todos os outros campos da tabela, seguindo a mesma estrutura, <propriedade><nome do campo><tipo><tamanho>.

O passo seguinte é inicializar alguns recursos do Framework DORM. Sendo assim, devem ser adicionas na cláusula Uses do formulário, as Untis a seguir:

```
dorm
dorm.adapters
dorm.Collections
dorm.Commons
dorm.Configuration
dorm.Loggers
dorm.Filters
```

Incluídas as Units, os recursos do Framework já podem ser carregados. Na declaração de variáveis do formulário principal (var), foi criada uma nova variável denominada Sessao do tipo TSession, presente na Unit dorm. A classe TSession é responsável por possibilitar a conexão com o banco de dados, oferecendo diversas funcionalidades para a manipulação dos objetos através do Framework, tais como as operações de CRUD (Create, Read, Update and Delete), dentre outras funcionalidades. Também foi declarada outra variável que recebeu a denominação de ListaPessoas e emprega o recurso de Generics, sendo do tipo TObjectList, de acordo com o trecho a seguir:

```
ListaPessoas: TObjectList<TPessoa>;
```

A classe TObjectList, presente em System.Generics.Collections, permite a criação de uma lista utilizando tipos genéricos, que neste caso é uma lista de objetos derivados da classe TPessoa.

Neste exemplo também será empregado o recurso de LiveBindings, sendo assim, foram adicionados ao formulário principal, um componente do tipo TBindScope e um do tipo TBindingList. O TBindScope permite que os dados de determinado componente possam ser acessados por outros componentes utilizando o recurso de LiveBindings, possibilitando a adição das expressões de ligação para os objetos visuais. Já o TBindingList tem o objetivo de possibilitar a elaboração de listas para as ligações. O objeto TBindScope recebeu a denominação de BindScopePessoa. Também foi adicionado um componente TListBox para que os nomes das pessoas cadastradas sejam exibidos.

Adicionados os objetos básicos para o emprego de LiveBindings, a próxima etapa é criar uma lista de expressões. Sendo assim, basta efetuar um clique duplo sobre o objeto BindingList1 para abrir a janela de edição de ligações. O próximo passo é clicar no botão New Binding para a criação de uma nova ligação. Com a mesma selecionada, através do Object Inspector, deve-se definir a propriedade ControlComponent com o valor de ListBox1, o componente que deve exibir as informações dos objetos, e a propriedade SourceComponent, o valor de BindScopePessoa, o componente de origem que contém os objetos. Em seguida, deve-se efetuar um clique duplo sobre o componente BindList1 para definir a expressão que será utilizada. Após selecionar a opção Format, presente em Collections, basta clicar no botão Add Expression. Com a nova expressão selecionada, em Control Expression, deve-se atribuir o valor Text, a propriedade do ListBox1 que exibirá as informações, e em Source Expression, o valor Current.Nome, para que seja mostrado o nome da pessoa atual. A **Figura 1** apresenta a janela para edição de ligações e a **Figura 2** demonstra a janela para edição das expressões.

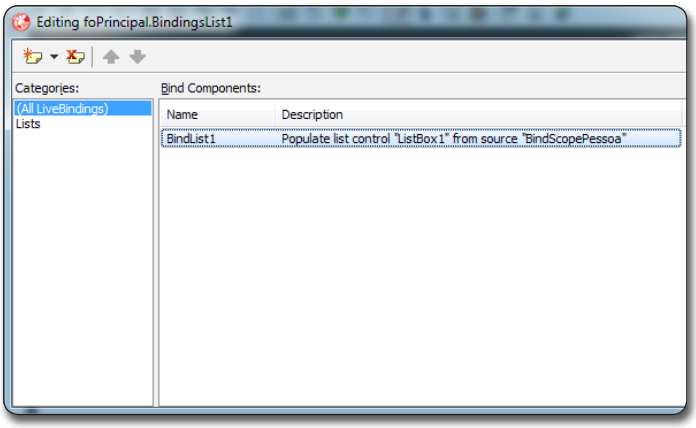


Figura 1. Janela de edição de ligações

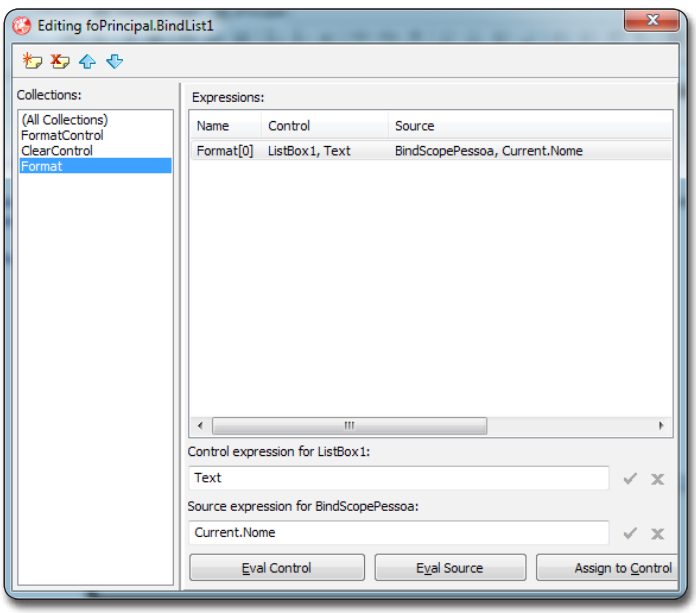


Figura 2. Janela de edição de expressões

Também foram adicionados ao formulário principal, três componentes do tipo TButton e um TEdit para futuras implementações.

A próxima etapa é iniciar os recursos do DORM, um processo que pode ser realizado no momento da criação do formulário. A **Listagem 5** demonstra o código atribuído ao evento OnCreate do formulário principal para a devido carregamento do Framework.

De início, são criadas duas variáveis do tipo string (linha 03) e posteriormente, elas recebem os nomes dos arquivos JSON definidos anteriormente (linhas 05 e 06), para que os mesmos possam ser carregados para a sessão. A sessão deve ser inicializada contendo como parâmetros, o arquivo base de configuração, o arquivo de

mapeamento e a definição da configuração atual do projeto (desenvolvimento, teste ou produção), neste caso de Development (desenvolvimento, conforme a linha 07). A seguir, a lista de pessoas é devidamente inicializada e o método FillList da sessão, que emprega Generics, é utilizado para carregar as informações do banco de dados e assim criar os objetos, que neste caso, são armazenados na lista de Pessoas (linhas 08 a 10). Na sequência, a propriedade DataObject do BindScopePessoa é utilizada para definir o objeto que será utilizado para a ligação, que neste caso é a lista de Pessoas, e o mesmo é ativado (linhas 11 e 12). Para concluir, o método FillList da lista de ligações BindList1, é chamado para efetuar a leitura e a exibição dos objetos no componente definido anteriormente, ListBox1 (linha 13).

Dando sequência ao projeto, a próxima etapa é implementar um simples código para possibilitar o cadastro de algumas informações. A **Listagem 6** apresenta o código atribuído ao evento OnClick do botão btNovo.

Listagem 5. Código do evento OnCreate do formulário principal

```
01 procedure TfoPrincipal.FormCreate(Sender: TObject);
02 var
03   ArquivoConf, Mapeamento: String;
04 begin
05   ArquivoConf := 'dormsqlite3.conf';
06   Mapeamento := 'mapeamento.conf';
07   Sessao := TSession.CreateConfigured(
     TStreamReader.Create(ArquivoConf),
     TStreamReader.Create(Mapeamento), deDevelopment);
08   ListaPessoas := TObjectList<TPessoa>.Create;
09   Sessao.InsertCollection(TObjectList<TObject>(ListaPessoas));
10   Sessao.FillList<TPessoa>(ListaPessoas);
11   BindScopePessoa.DataObject := ListaPessoas;
12   BindScopePessoa.Active := True;
13   BindList1.FillList;
14 end;
```

Listagem 6. Código do evento OnClick do botão btNovo

```
01 procedure TfoPrincipal.btNovoClick(Sender: TObject);
02 var
03   NovaPessoa: TPessoa;
04   DadosPessoa: TArray<String>;
05 begin
06   SetLength(DadosPessoa, 2);
07   if InputQuery('Nova Pessoa', ['Nome', 'Endereco'], DadosPessoa) then
08   begin
09     NovaPessoa := TPessoa.Create(DadosPessoa[0], DadosPessoa[1]);
10     ListaPessoas.Add(NovaPessoa);
11     Sessao.Persist(NovaPessoa);
12     Sessao.FillList<TPessoa>(ListaPessoas);
13     with BindScopePessoa do
14     begin
15       Active := False;
16       DataObject := ListaPessoas;
17       Active := True;
18     end;
19     BindList1.FillList;
20     ListBox1.ItemIndex := ListBox1.Items.Count - 1;
21   end;
22 end;
```

Foram declaradas duas variáveis, uma do tipo TPessoa para auxiliar no processo de criação da pessoa, e a outra do tipo TArray (linhas 03 e 04). Inicialmente, é definido o tamanho do array de strings para dois através do método SetLength (linha 06). Em seguida, é utilizada a função InputQuery (linha 07) para carregar as informações da nova pessoa, nome e endereço, de uma só vez e armazenar os valores das mesmas no array, DadosPessoas. Ao confirmar o cadastro, é criado um objeto do tipo TPessoa, sendo o mesmo preenchido com as informações contidas em DadosPessoa (linha 09). Posteriormente, a nova pessoa é adicionada ao objeto ListaPessoas através do método Add e as informações do novo objeto são gravadas no banco de dados através do método Persist da sessão, lembrando que o método Save também pode ser utilizado (linhas 10 e 11). Em seguida, a lista de pessoas é novamente preenchida, bem como o objeto BindScopePessoas (linhas 12 a 18). Para concluir, a lista de objetos é novamente exibida no ListBox1 através do método FillList e o último item do ListBox1 é selecionado através da propriedade ItemIndex (linhas 19 e 20).

O próximo passo é implementar o código para o botão de exclusão de uma pessoa. A **Listagem 7** demonstra o código adicionado no evento OnClick do botão btDeletar.

Listagem 7. Código do evento OnClick do botão btDeletar

```
01 procedure TfoPrincipal.btDeletarClick(Sender: TObject);
02 var
03   Pessoa: TPessoa;
04 begin
05   if ListBox1.ItemIndex >= 0 then
06   begin
07     Pessoa := ListaPessoas[ListBox1.ItemIndex];
08     Sessao.Delete(ListaPessoas.Extract(Pessoa));
09     BindList1.FillList;
10   end;
11 end;
```

Neste trecho, antes de efetuar a exclusão, é verificado se alguma pessoa está selecionada no ListBox1 através da propriedade ItemIndex (linha 05), e caso esteja, então é utilizada uma variável do tipo TPessoa temporariamente para receber a pessoa atual de acordo com a seleção do ListBox1 (linha 07). Em seguida, o método Delete do objeto Sessao é utilizado para remover a Pessoa selecionada (linha 08). Para concluir, é efetuada a atualização da exibição das pessoas (linha 09).

A próxima etapa deste projeto é implementar o recurso de pesquisa, que funciona semelhante a um filtro. No DORM, este recurso pode ser obtido através da utilização da interface ICriteria, presente em dorm.Filters. A **Listagem 8** demonstra como implementar uma pesquisa através deste Framework utilizando o evento OnClick do botão btPesquisar.

Após desativar o BindScopePessoa (linha 05), é verificado se o objeto TEdit que deve conter o nome da pessoa que será pesquisada está vazio, e caso tal sentença seja verdadeira, simplesmente é efetuado o carregamento da lista de pessoas sem nenhum filtro (linhas 06 e 07), caso contrário, é criado novo critério para ser

empregado na busca através do método NewCriteria (linha 10). Este método recebe o nome do campo que deve ser pesquisado, o tipo da pesquisa (neste caso igual) e o valor para a comparação. Para aplicar o critério, basta simplesmente utilizar o mesmo método FillList da Sessao informando o objeto ListaPessoas em conjunto com o mesmo (linha 11). Para concluir, o objeto BindScopePessoa é novamente carregado e atualizado, onde posteriormente as informações filtradas são atualizadas no ListBox1 (linhas 13 a 15).

O DORM possui diversos operadores de comparação que podem ser utilizados para efetuar um filtro ou pesquisa. A **Tabela 1** apresenta alguns destes operadores.

Listagem 8. Código do evento OnClick do botão btPesquisar

```
01 procedure TfoPrincipal.btPesquisarClick(Sender: TObject);
02 var
03   Criterio: ICriteria;
04 begin
05   BindScopePessoa.Active := False;
06   if edPesquisa.Text = '' then
07     Sessao.FillList<TPessoa>(ListaPessoas)
08   else
09   begin
10     Criterio := TdormCriteria.NewCriteria('Nome',
       TdormCompareOperator.coEqual, edPesquisa.Text);
11     Sessao.FillList<TPessoa>(ListaPessoas, Criterio);
12   end;
13   BindScopePessoa.DataObject := ListaPessoas;
14   BindScopePessoa.Active := True;
15   BindList1.FillList;
16 end;
```

Comparador	Descrição
coEqual	Para localizar informações idênticas as fornecidas.
coGreatherThan	Localizar informações maiores que o valor fornecido.
coLowerThan	Localizar informações menores que o valor.
coGreatherOrEqual	Maior ou igual ao valor indicado.
coLowerOrEqual	Menor ou igual ao valor indicado.
coNotEqual	Localizar informações diferentes do valor.
coLike	Localizar informações que contenham determinado valor.

Tabela 1. Operadores de comparação para pesquisas através do DORM

É possível notar que o DORM oferece diversas funcionalidades e, como apresentado neste exemplo, várias operações podem ser realizadas sem que exista a necessidade de empregar uma linha sequer de instruções SQL.

Para concluir o projeto, no evento OnDestroy do formulário, basta liberar a sessão da memória, através de Sesssao.Free, bem como a lista de pessoas (ListaPessoa.Free).

Conclusão

Atualmente, empregar os vários recursos da Orientação a Objetos é, sem dúvida, um fator de extrema importância, uma vez que suas metodologias trazem grandes benefícios. Além do mais, o Delphi atualmente conta com uma diversidade de recursos nativos e de terceiros que possibilitam o emprego de OO rapidamente e de forma simples. Como apresentado, empregar ORM no Delphi é uma tarefa facilitada com a utilização do DORM, que por sua vez é um Framework estável, de código fonte aberto e que traz consigo diversas funcionalidades, abstraindo a camada de conexão com o banco de dados e suas tabelas, o que evidentemente, permite um ganho na produtividade.

Entretanto, existem alguns recursos que ainda não estão disponíveis no Framework, tal como o mapeamento automático de entidades e atributos, de forma semelhante ao recurso de Annotations em Java, um fator que facilitaria ainda mais o emprego do DORM, mas que provavelmente deverá ser implementado futuramente.

De qualquer forma, seja através do DORM ou outro Framework, o emprego de ORM através do Delphi é uma excelente prática para os projetos, possibilitando que os recursos de OO possam ser usufruídos, sem contar que as mesmas se tornam melhores planejadas e de fácil manutenção.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Maiores informações e download do DORM
http://code.google.com/p/delphi-orm/

Maiores informações sobre o SQLite e download de algumas ferramentas
http://www.sqlite.org/cvstrac/wiki?p=ManagementTools

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Análise de código

Desvendando métricas de qualidade e estimativas

Todo desenvolvedor possui uma noção do quanto um código escrito por ele está bem escrito ou não. Contudo, em um projeto grande, onde muitas mãos são utilizadas, isso foge ao controle.

Para auxiliar nesse problema vários softwares de análise de código foram construídos, todos mostram diversas métricas. Mas muitas não são compreendidas. Dentre elas a que é menos entendida é a complexidade ciclomática.

Complexidade Ciclomática: conheça para evitar

Essencialmente, com complexidade ciclomática números altos são ruins e números baixos são bons. Essa métrica é utilizada para medir quão difícil será testar, manter ou ajustar um determinado código, bem como indicar que esse código pode produzir erros. De uma maneira simples o valor dessa métrica é obtido ao contar o número de decisões feitas em uma parte do código.

Realizando uma medição

A ferramenta Pascal Analyzer (veja seção **Links**) oferece uma métrica similar à complexidade ciclomática. Depois de instalada, através de um relatório de complexidade é possível visualizar a quantidade de pontos de decisão de um código. Assim, foi criado um projeto do tipo console para demonstrar a evolução da complexidade. Automaticamente foi adicionado a esse projeto um bloco try/except que deve ser excluído, permanecendo apenas o begin/end, vazios.

Ao abrir o Pascal Analyzer é necessário criar um novo projeto. Isso é feito ao acessar o menu File>New Project, que exibe a janela da **Figura 1**.

No campo Main file informamos o arquivo .dpr do nosso projeto. Além disso delimitamos o campo de pesquisa da ferramenta selecionando a opção Main file. Isso determina que apenas o projeto principal será analisado e não o restante de código relacionado (por exemplo, as units do próprio Delphi). O Pascal Anlyzer disponibiliza uma série de relatórios, mas o foco é a medição dos pontos de decisão. Na guia Reports da janela do projeto é possível configurar o que analisar. No caso, ficará habilitado somente a opção de totalização, que exibe um resumo geral, e complexidade (**Figura 2**).

Com tudo isso configurado e confirmado, basta clicar no menu Analysis>Run. O relatório da **Figura 3** é exibido.

Resumo DevMan

Analisando código:

A indústria de software sempre cria novos padrões de qualidade que são explorados pelas mais diversas ferramentas, através de métricas, que sugerem que ao serem usadas, um código poderá ser melhorado em qualidade. Contudo, de nada basta um relatório repleto de números se esses não podem ser interpretados pelo desenvolvedor. O mercado também exige prazo de entrega das tarefas a serem implementadas. É sempre um desafio definir essa estimativa de prazo, mas existe outro tipo de métrica que pode ser utilizada, a chamada análise de pontos de função (APF). Neste artigo o fundamento de algumas métricas de qualidade é abordado de forma prática e direta, para que o desenvolvedor não tenha dúvidas de sua valia e funcionalidade e todo um passo a passo para se calcular prazo usando APF é mostrado.

Em que situação o tema é útil:

Quando se deseja aplicar uma análise de qualidade no código, para direcionar medidas de correção e melhorias e mensurar o tempo de desenvolvimento de uma solução ou funcionalidade.

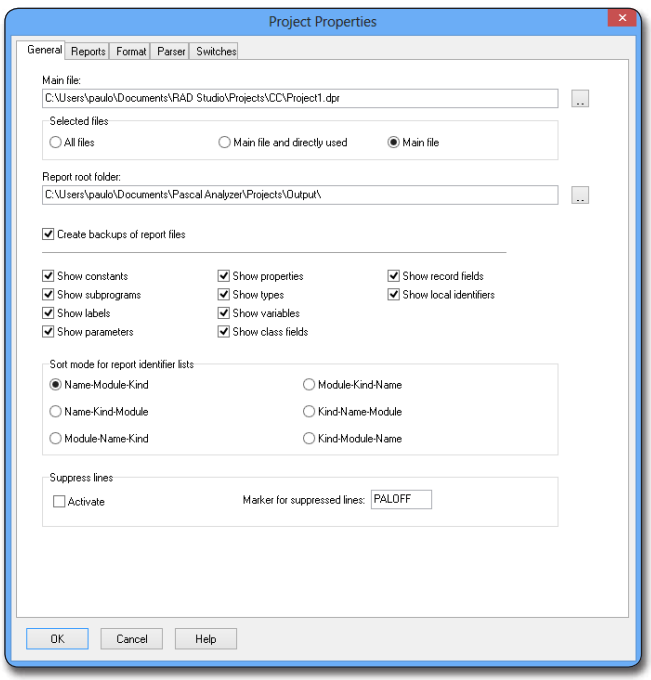


Figura 1. Novo projeto

Nele foi destacado o resultado a análise. Em um primeiro momento, nada foi identificado, não temos complexidade alguma. Contudo, sem fechar o Pascal Analyzer, modificamos o código do projeto para que fique como a seguir e salvamos o projeto:

```
var
 x: Integer;
begin
 x:= 5;
 if x=4 then
  Write('Primeiro teste');
end.
```

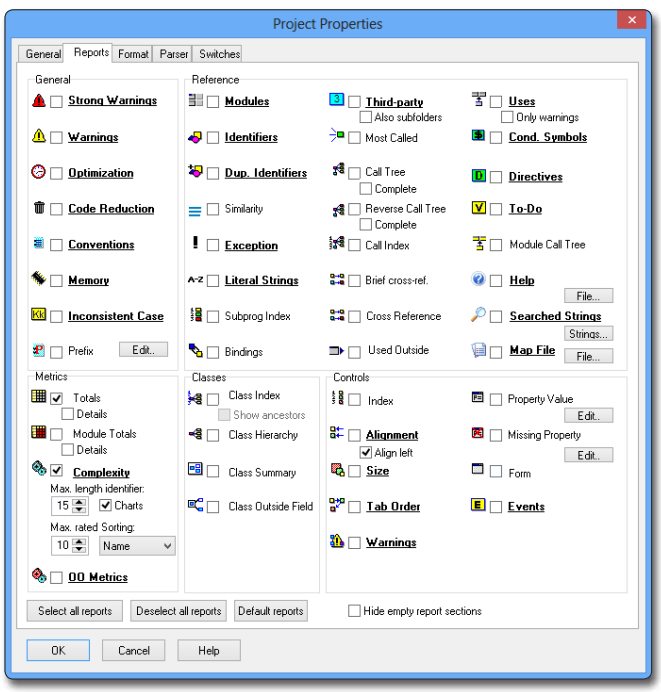


Figura 2. Opções de relatórios

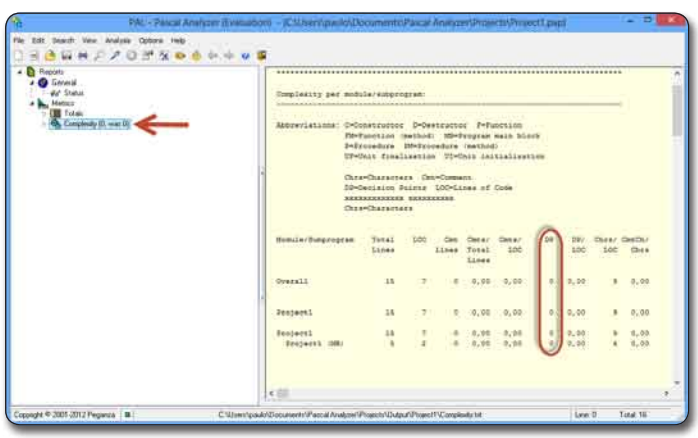


Figura 3. Resultado da primeira análise

Ao executar novamente o código, temos um resultado diferente (**Figura 4**).

Um simples If aumentou a complexidade para dois pontos. Imagine agora ao trocar o código do projeto pelo a seguir:

```
var
 x: Integer;
begin
 x:= 5;
 case x of
   1: Write ('Opção 1');
   2: Write ('Opção 2');
   3: Write ('Opção 3');
   4: Write ('Opção 4');
   5: Write ('Opção 5');
 end;
end.
```

A complexidade sobe para 7 pontos (**Figura 5**).

Module/Subprogram	Total	LOC	Cmt	Cmts/	Cmts/	DP	DP/	Chrs/	CmtCh/
	xxxxx		xxxxx	xxxxx Lines	xxx		xxx	xxx	xxxx
Overall	18	10	0	0,00	0,00	2	0,20	10	0,00
xxxxxxxx	xx	xx	x	xxxx	xxxx	x	xxxx	xx	xxxx
Project1	18	10	0	0,00	0,00	2	0,20	10	0,00
Project1 (MB)	6	5	0	0,00	0,00	2	0,40	10	0,00

Figura 4. Segunda análise

Module/Subprogram	Total Lines	LOC	Cmt Lines	Cmts/ Total Lines	Cmts/ LOC	DP	DP/ LOC	Chrs/ LOC	CmtCh/ Chrs
xxxxxxx	xx	xx	x	xxxx	xxxx	x	xxxx	xx	xxxx
Project1	22	15	0	0,00	0,00	7	0,47	12	0,00
Project1	22	15	0	0,00	0,00	7	0,47	12	0,00
Project1 (MB)	10	10	0	0,00	0,00	7	0,70	13	0,00

Figura 5. Terceira análise

Existe um valor ideal?

Assim como muitas métricas existentes na indústria, não existe um limite que se encaixe nas organizações como o ideal. Contudo, as boas práticas, ditadas pela própria indústria de software, estipulam que um limite de 10 é um bom começo. Inclusive esse valor foi proposto por McCabe, o criador da métrica da complexidade ciclomática.

Outras empresas utilizam um valor limite de 15, que também reporta bons resultados. Vale lembrar que esse número é por método, procedimento, função e não do projeto todo.

E quanto à quantidade de linhas?

Existe uma relação entre a quantidade de linhas de um procedimento e a qualidade de código. Quanto mais linhas uma função possuir, mais suscetível a erros ela será. Contudo, ao combinar a análise de complexidade ciclomática com a quantidade de linhas de código, isso fica mais visível. Isso foi comprovado em pesquisas da NASA, em seu Centro de Tecnologia de Garantia de Software.

LOC: contagem de linhas de código

Essa é uma métrica muito utilizada para se mensurar o tamanho de projeto, visto que ela conta a quantidade de linhas de código. Esse é seu objetivo. Essa quantidade pode ser obtida analisando

linhas lógicas ou linhas físicas. A contagem lógica tenta analisar o código que foi quebrado em linhas, talvez para uma melhor visualização, e ignorar essa quebra quando necessário. Por exemplo, o seguinte trecho de código seria tratado como uma única linha:

```
If (variavel = true) then
Begin
  Inc(contador);
End;
```

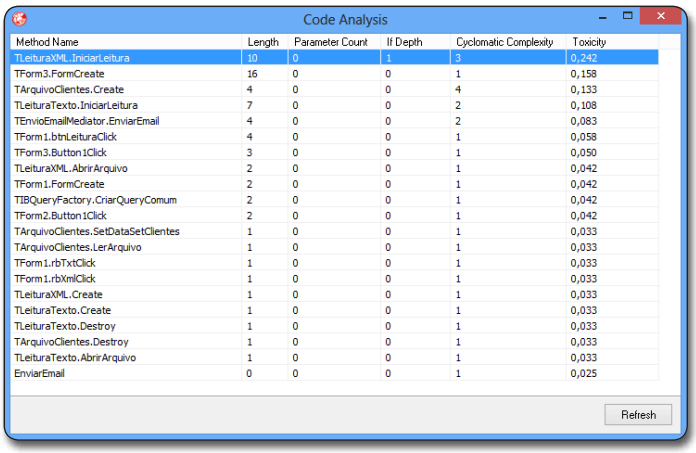
A quebra de linha após a palavra reservada then e o uso do Begin/End são desnecessários nesse cenário, é justamente isso que o analisador tenta resolver. Já a contagem física iria retornar uma conta de quatro linhas. Ao comparar as duas, a contagem lógica retorna algo muito mais "real", removendo o que não é necessário. A grande discussão que existe quanto ao uso dessa métrica é: seu uso para medir produtividade.

A LOC não leva em conta a experiência de quem desenvolveu o código, nem muito menos a qualidade do mesmo. É uma situação muito comum, desenvolvedores mais experientes criam código mais otimizado e de menor tamanho do que desenvolvedores júniores. Se essa métrica for utilizada como a única forma de medição de produtividade, pode gerar resultados controversos e até mesmo estimular a produção de código de baixa qualidade. A **Figura 6** exibe a janela de análise de código gerada pela ferramenta Castalia.

O Castalia (ver seção **Links**) é um Addin para o IDE do Delphi que implementa diversos recursos, como refatoração aprimorada, marcação visual de início e fim de bloco de comandos e análise de código. Essa análise exibe o resultado de algumas métricas, sendo o primeiro o de LOC, contudo, a contagem feita é a lógica e por método. Para exemplificar isso observar a linha do método TEnvioEmailMediator.EnviarEmail. Em sua contagem foram identificadas quatro linhas, mas, ao abrirmos o código fonte encontramos o que mostra o código a seguir:

```
procedure TEnvioEmailMediator.EnviarEmail(cds: TClientDataSet);
begin
  cds.First;
  while not cds.Eof do
  begin
    MediatorU.EnviarEmail(cds.FieldByName('email').AsString);
    cds.Next;
  end;
end;
```

Se a linha de declaração do método for ignorada temos aí oito linhas, uma diferença e tanto. Mas ao pensar um pouco mais, entendemos como a contagem de apenas quatro se deu. Os comandos Begin/End são ignorados da contagem, pois, tecnicamente são apenas indicadores de blocos, e que não desempenham nenhum papel real/funcional no método. Ao observar novamente a **Figura 6** é clara a visualização de uma métrica chamada Toxicity. Esta é resultado de um cálculo específico que combina os resultados das outras métricas, gerando um valor único que quantifica quão tóxico a o método é. Segundo os desenvolvedores do Castalia, métodos com valores maiores que 1.0 são grandes candidatos a uma boa revisão.



Method Name	Length	Parameter Count	If Depth	Cyclomatic Complexity	Toxicity
TLeituraXML.IniciarLeitura	10	0	1	3	0,242
TForm3.FormCreate	16	0	0	1	0,158
TArquivoClientes.Create	4	0	0	4	0,133
TLeituraTexto.IniciarLeitura	7	0	0	2	0,108
TEnvioEmailMediator.EnviarEmail	4	0	0	2	0,083
TForm1.btnLeituraClick	4	0	0	1	0,058
TForm3.Button1Click	3	0	0	1	0,050
TLeituraXML.AbrirArquivo	2	0	0	1	0,042
TForm1.FormCreate	2	0	0	1	0,042
TIBQueryFactory.CriarQueryComum	2	0	0	1	0,042
TForm2.Button1Click	2	0	0	1	0,042
TArquivoClientes.SetDataSetClientes	1	0	0	1	0,033
TArquivoClientes.LerArquivo	1	0	0	1	0,033
TForm1.rbTxtClick	1	0	0	1	0,033
TForm1.rbXmlClick	1	0	0	1	0,033
TLeituraXML.Create	1	0	0	1	0,033
TLeituraTexto.Create	1	0	0	1	0,033
TLeituraTexto.Destroy	1	0	0	1	0,033
TArquivoClientes.Destroy	1	0	0	1	0,033
TLeituraTexto.AbrirArquivo	1	0	0	1	0,033
EnviarEmail	0	0	0	1	0,025

Figura 6. Janela de análise de código do Castalia

Comment Radio

Existe um fato que é verdadeiro entre os programadores, se não for obrigado, muitos não comentam seus códigos. Essa é uma atitude até mesmo natural de quem faz o código, porém, quando se tem um ambiente onde muitos estão envolvidos, e conhecimento precisa ser compartilhado. Uma forma de distribuir esse conhecimento é através de comentários. Assim, assume-se que um código com baixa quantidade de comentários pode não ser considerado de alta qualidade. No Delphi XE 3 temos no menu Project a opção QA Metrics. Essa opção exibe uma janela (**Figura 7**) com vários tipos de métricas que podem ser aplicadas ao projeto que está aberto no momento.

Cada métrica é explicada por uma descrição e seus valores base de comparação ainda podem ser configurados. Por exemplo, na **Figura 7** a única métrica que será extraída é a Comment Ratio. O resultado de sua aplicação em um projeto de teste é exibido na **Figura 8**.

Como é possível ver, várias partes do código estão sem comentário, o Comment Ratio (CR) está zero. Mas é preciso tomar cuidado também com os códigos que mostram algum valor de CR. Se o método EnviarEmail, que possui um valor de 20, for aberto, é possível ver que o comentário existente nada mais é do que o padrão criado pelo Delphi, como é mostrado a seguir:

```
{ TEnvioEmailMediator }
procedure TEnvioEmailMediator.EnviarEmail(cds: TClientDataSet);
```

Métricas para estimativa

A métrica mais conhecida para estimativa de tempo é a análise de pontos de função. Com seu uso pode-se estimar o tamanho das funcionalidades de um sistema e, em função disso, definir o tempo e recursos para um projeto em questão com base na produtividade da equipe ou na taxa de entrega de projetos semelhantes. Quando essas estimativas são definidas no início do projeto fica fácil planejar interações, criar um orçamento, pensar no custo e assim evitar ao máximo, riscos não planejados. Não que eles não venham a ocorrer, mas, com um planejamento é muito possível encerrar um projeto dentro de prazo e custo previstos.

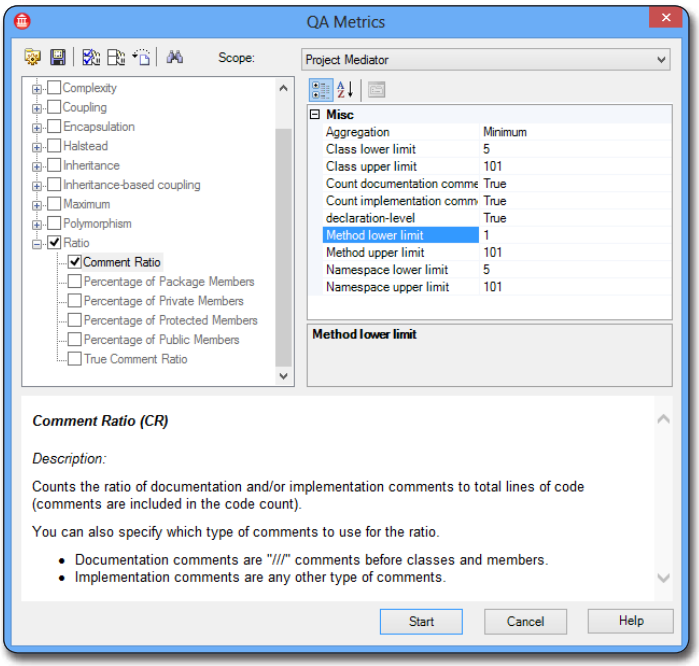


Figura 7. QA Metrics

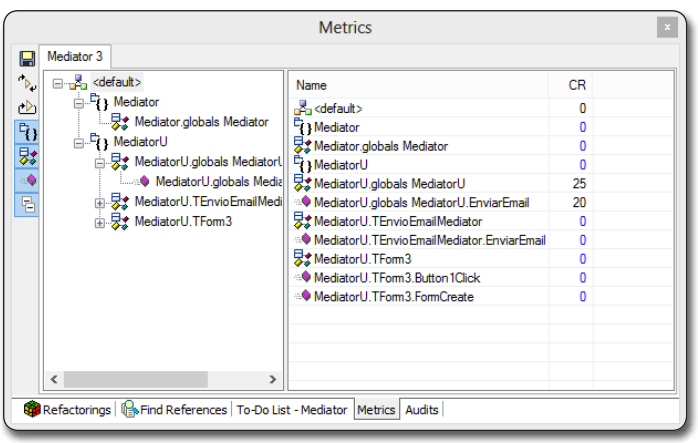


Figura 8. Resultado da métrica Comment Ratio

Contando

A contagem de pontos de função é realizada através de um conjunto de passos previamente determinados. Para compreender o funcionamento na prática, vamos imaginar um conjunto de requisitos e então realizar o cálculo.

Temos um sistema de controle de edições de revistas. Nele são armazenadas as edições da Clube Delphi, seus artigos e autores.

Esse sistema ao ser executado exibe uma tela de pesquisa de artigos (**Figura 9**), onde informamos uma palavra chave e a mesma será buscada. Ao efetuar um duplo clique sobre algum artigo localizado, é exibida sua tela de edição. Ao selecionar um registro e pressionar a tecla delete, o mesmo é removido.

O menu cadastros oferece meios para se cadastrar autores (código, nome, e-mail), além é claro dos artigos (código, edição, título, palavras-chave, autor). O cadastro de autores é feito a partir da tela de consulta de autor (**Figura 10**).

A **Figura 11** mostra o cadastro de autores, enquanto que a **Figura 12** o cadastro de artigos.

Qual é o tempo estimado para concluir esse pequeno projeto? Antes de responder essa questão é preciso mensurar o tamanho da aplicação, isso é feito através da contagem de pontos de função. Mas para aplicar essa contagem, é preciso seguir alguns passos:

1 – Definir o tipo de contagem;
2 – Definir fronteira da aplicação e escopo da contagem;
3 – Calcular funções de dados;
4 – Calcular funções de transação;
5 – Calcular pontos de função não ajustados;
6 – Calcular valor do fator de ajustes;
7 – Calcular pontos de função ajustados.

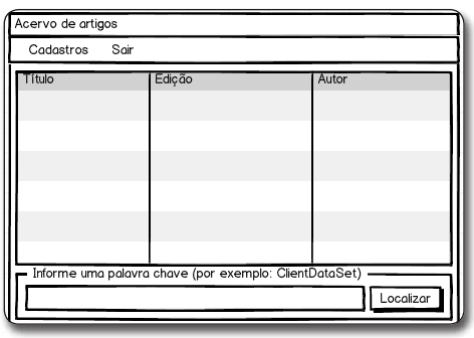


Figura 9. Busca de edições

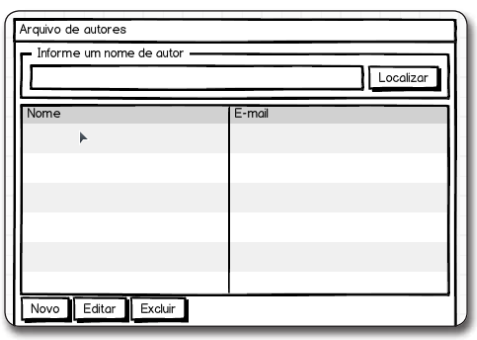


Figura 10. Consulta de autores

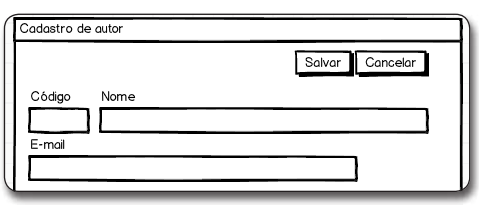


Figura 11. Cadastro de autor

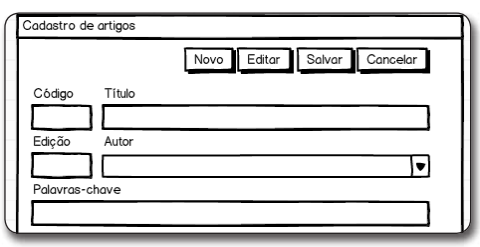


Figura 12. Cadastro de artigos

Cada passo é revisto a seguir:

1) Tipo de Contagem: existem três tipos de contagem de pontos de função. A diferença no procedimento adotado entre esses tipos de contagem está nas fórmulas aplicadas na contagem. São eles:

• Projeto de desenvolvimento: mede todas as funções que serão entregues com o projeto em sua primeira versão. É o tipo de contagem que será utilizada neste artigo;

• Projeto de melhoria: mede as funcionalidades alteradas, incluídas e excluídas ao projeto;

• Aplicação: mede as funções de uma aplicação já instalada.

2) Escopo da Contagem: é a fronteira da aplicação. Define as funções que serão incluídas em uma determinada contagem de pontos de função. Como estamos criando um acervo de artigos, as funcionalidades são as que foram exibidas anteriormente;

3) Funções de Dados: consiste na contagem dos tipos de dados utilizados nas funcionalidades a serem desenvolvidas. Esses tipos de dados podem ser agrupados em um Arquivo Lógico Interno (ALI) que representa grupos de dados relacionados e reconhecidos pelo usuário, dentro da fronteira da aplicação; ou um Arquivo de Interface Externa (AIE) que representa dados referenciados pela aplicação, mas mantidos dentro da fronteira de outra aplicação. No exemplo temos apenas o grupo ALI, composto por Artigo e Autor. Para calcular a complexidade das funções de dados, deve-se contar o número de Tipos de Dados (TD) e Tipos de Registros (TR) de cada uma dessas funções. Um TD refere-se a um campo único reconhecido pelo usuário, sem repetição, ou seja, se um campo se repetir mais de uma vez numa interface, conta-se apenas uma vez. Um TR refere-se a um subgrupo de tipos de dados, também reconhecidos pelo usuário e componente de um ALI ou AIE.

Dessa forma, a **Tabela 1** exibe os tipos de dados Autor e a **Tabela 2**, de Artigo.

Para entender o que é o Tipo de Registro, vamos imaginar que artigo fosse categorizado, por exemplo, em "teóricos" e "práticos", artigo teria valor dois para TR. Já autor, que não possui categorias, possui valor um.

Identificados os TDs e TRs, pode-se calcular a complexidade de cada ALI/AIE através da **Tabela 3**. Analisando-a, observa-se que o ALI Autor é de complexidade baixa, pois possui um TRs e três TDs, o ALI Artigo também é de complexidade baixa, pois possui um TR e 5 TDs.

4) Funções de Transação: representam os processos elementares fornecidos pela aplicação ao usuário, onde um processo elementar identifica a menor unidade funcional do ponto de vista do usuário, e pode ser:

• Entrada Externa (EE): para sua identificação devem ser analisados todos os processos elementares que processam dados vindos de fora da fronteira da aplicação e que atualizam um ou mais ALIs;

• Saída Externa (SE): o processo gera dados para fora da fronteira da aplicação, tendo por objetivo principal apresentar dados ao usuário através de lógica de processamento que não apenas a recuperação de dados;

• Consulta Externa (CE): processo que envia dados para fora da fronteira da aplicação, apresentando dados ao usuário por meio de uma simples recuperação de informações de ALI ou AIE.

A **Tabela 4** exibe a identificação dos processos de projeto de exemplo.

As regras para definir a complexidade de EE, SE ou CE consideram o número de Arquivos Referenciados (AR) e o número de Tipo de Dados (TD) utilizados, sendo:

• AR: Para cada ALI lido ou mantido pela aplicação, ou AIE lido, conta-se um;

• TD:

- Entrada Externa (EE): contam-se os atributos que são atualizados e determina-se a complexidade conforme a **Tabela 5**;

- Saída Externa (SE): contam-se os atributos da saída, além de atributos calculados, e determina-se a complexidade conforme **Tabela 6**;

TD	Tipo de Dado
1	Código do autor
2	Nome do autor
3	E-mail do autor

Tabela 1. Tipos de Dados de Autor

TD	Tipo de Dado
1	Código do artigo
2	Título do artigo
3	Edição do artigo
4	Autor do artigo
5	Palavras chave do artigo

Tabela 2. Tipos de Dados de Artigo

Tipos de Registro (TR)	Tipos de Dados (TD)		
	Abaixo de 20	20 a 50	Acima de 50
1	Baixa	Baixa	Média
2 a 5	Baixa	Média	Alta
Acima de 5	Média	Alta	Alta

Tabela 3. Complexidade das funções de dados (ALI e AIE)

Funcionalidade	Processo Elementar	Tipo
Pesquisa de Artigos	Listar Artigos	CE
	Pesquisar Artigos	CE
	Editar Artigos	EE
	Excluir Artigos	EE
Cadastramento Artigos	Incluir Artigo	EE
Pesquisa de Autores	Listar Autores	CE
	Pesquisar Autores	CE
	Incluir Autor	EE
	Alterar Autor	EE
	Excluir Autor	EE

Tabela 4. Processos Elementares identificados no projeto de exemplo

Arquivos Referenciados (AR)	Tipos de Dados (TD)		
	Abaixo de 5	5 a 15	Acima de 15
0 ou 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
Acima de 2	Média	Alta	Alta

Tabela 5. Complexidade de Entrada Externa (EE)

Arquivos Referenciados (AR)	Tipos de Dados (TD)		
	Abaixo de 6	6 a 19	Acima de 19
0 ou 1	Baixa	Baixa	Média
2 ou 3	Baixa	Média	Alta
Acima de 3	Média	Alta	Alta

Tabela 6. Complexidade de Saída Externa (SE) e Consulta Externa (CE)

- Consulta Externa (CE): em um processo de entrada de dados, contam-se os atributos de seleção, além de mensagem ao usuário quando for o caso. Em um processo de saída de dados, contam-se os atributos de saída. Em processos de entrada e saída, somam-se os dois resultados. Determina-se a complexidade conforme a **Tabela 6**.

Aplicando essas regras é possível mensurar a complexidade dos processos do projeto, como mostra a **Tabela 7**.

Processo Elementar	Tipo	AR	TD	Complexidade
Listar Artigos	CE	1	3	Baixa
Pesquisar Artigo	CE	1	4	Baixa
Incluir Artigo	EE	2	5	Média
Alterar Artigo	EE	2	5	Média
Excluir Artigo	EE	2	5	Média
Consultar Autor	CE	1	3	Baixa
Listar Autor	CE	1	2	Baixa
Incluir Autor	EE	1	3	Baixa
Alterar Autor	EE	1	3	Baixa
Excluir Autor	EE	1	3	Baixa

Tabela 7. Complexidade dos processos elementares

O processo Listar Artigos possui TD igual a três porque esse é o número de campos que são exibidos no grid, já Pesquisar Artigo possui 4, pois inclui o campo de pesquisa. Já para os processos Incluir, Alterar e Excluir Artigo, o valor de AR é dois porque eles envolvem duas entidades, o valor de duas entidades é utilizado Artigo e Autor.

5) Pontos de Função Não Ajustados: após identificar as funções de dados e os processos elementares, multiplica-se o total de ALI, AIE, EE, SE e CE pelo respectivo valor de ponto de função da tabela de complexidade (**Tabela 8**) para determinar o valor total que será o PF não ajustado.

A **Tabela 9** exibe o resultado final do cálculo de pontos de função não ajustados para o projeto de acervo de artigos.

Função	Baixa	Média	Alta
Consulta Externa (CE)	3	4	6
Entrada Externa (EE)	3	4	6
Saída Externa (SE)	4	5	7
Arquivo de Interface Externa (AIE)	5	7	10
Arquivo Lógico Interno (ALI)	7	10	15

Tabela 8. Contribuição de ponto de função por complexidade

Processo Elementar	Tipo	Complexidade	Pontos Função
Artigo	ALI	Baixa	7
Autor	ALI	Baixa	7
Listar Artigos	CE	Baixa	3
Pesquisar Artigos	CE	Baixa	3
Incluir Artigo	EE	Média	4
Alterar Artigo	EE	Média	4
Excluir Artigo	EE	Média	4
Listar Autor	CE	Baixa	3
Pesquisar Autor	CE	Baixa	3
Incluir Autor	EE	Baixa	3
Alterar Autor	EE	Baixa	3
Excluir Autor	EE	Baixa	3
Total de Pontos de Função			47

Tabela 9. Identificação dos processos elementares e suas complexidades

6) Valor do Fator de Ajuste: representa a influência de requisitos técnicos e de qualidade no tamanho do software. É calculado com base em 14 características gerais de um sistema, onde cada uma delas deve ser analisada com relação ao seu nível de influência sobre o sistema e pontuada de 0 (nenhuma influência) a 5 (grande influência).

A descrição completa de cada característica e as tabelas para avaliação dos valores de pontuação pode ser obtida no manual de Práticas de Contagem de Pontos de Função, disponibilizado pelo grupo internacional de usuários de pontos de função (ver seção **Links**) - IFPUG. As características foram descritas a seguir, bem como os valores considerados para cada uma delas:

1. Comunicação de Dados: considera se são utilizados recursos de comunicação pela aplicação. Neste exemplo, será considerado o valor quatro, que é relativo a aplicações on-line, suportadas por algum protocolo de comunicação;
2. Processamento Distribuído: relativo à transferência de dados entre os componentes da aplicação. Foi considerado o valor um, uma vez que não realiza processamento distribuído, fazendo uso apenas de um sistema de gerenciamento de banco de dados;
3. Performance: determina como o tempo de resposta influencia a aplicação. Foi considerado zero, uma vez que não foi indicada nenhuma restrição a esse respeito;
4. Configuração Altamente Utilizada: define o nível de restrições impostas pelo usuário. Também foi considerado o valor zero uma vez que nenhuma restrição foi imposta;

5. Volume de Transações: refere-se ao volume de informações processados pela aplicação. Considerou-se o valor um, uma vez que é previsto um número de transações mais elevado em alguns períodos específicos, como a cada período de matrículas de alunos;
6. Entrada de Dados On-line: considera a quantidade de transações feitas on-line. Foi considerado o valor cinco uma vez que o sistema apresenta todas as transações como sendo desse tipo;
7. Eficiência do Usuário Final: refere-se a facilidades oferecidas ao usuário final, como ajuda on-line, auxílio à navegação por teclas de função, menus, dentre outros. Foi considerado o valor um por apresentar algumas poucas características desse tipo, como a exibição dos cursos em caixa de combinação para facilitar a seleção do usuário no cadastramento de alunos;
8. Atualização On-line: define se os arquivos lógicos internos são atualizados on-line. Foi utilizado o valor três, pois todos os arquivos internos são atualizados dessa forma;
9. Complexidade de Processamento: determina a complexidade de processamento das funcionalidades da aplicação. Foi considerado o valor zero, por ser uma aplicação bastante simples;
10. Reutilização: identifica se o código foi projetado para ser reaproveitado em outras aplicações. O valor zero foi considerado por essa não ter sido uma preocupação no projeto;
11. Facilidade de Instalação: considera se existem ferramentas de conversão e instalação da aplicação. Também foi considerado o valor zero por essa preocupação não ter sido levada em consideração no estudo de caso;
12. Facilidade de Operação: considera aspectos de segurança e recuperação de informações. Como isso também não foi considerado, é assumido o valor zero;
13. Múltiplos Locais: se a aplicação foi projetada e desenvolvida para ser utilizada em diferentes locais. Essa restrição também não foi considerada e assumido o valor zero;
14. Facilidade de Mudanças: se a aplicação foi projetada para facilitar mudanças na lógica de processamento ou em suas estruturas de dados. O valor zero foi utilizado nesse caso.

Somam-se então essas pontuações para obter o nível total de influência (TDI – Total Degree of Influence). Daí basta aplicar a seguinte fórmula para obter o valor do fator de ajuste (VAF – Value Adjustment Factor):

Valor do Fator de Ajuste (VAF) = (TDI x 0,01) + 0,65

Atualmente esse é um passo opcional do processo de contagem. Muitas organizações desconsideram o fator de ajuste e usam apenas a medição dos pontos de função não ajustados. Isso se deve ao fato de que as características gerais de um sistema, apresentadas anteriormente, são consideradas desatualizadas ou incompletas em relação às tecnologias atualmente utilizadas, fazendo com que sua utilidade seja questionada.

7) Pontos de Função Ajustados: Os PFs ajustados são calculados a partir dos PFs não ajustados multiplicado pelo fator de ajuste. No caso de processos de desenvolvimento, funções para conversão de dados também devem ser consideradas e sua complexidade em pontos de função deve ser adicionada aos pontos de função não ajustados. Assim, a seguinte fórmula para processos de desenvolvimento é aplicada:

PF Desenvolvimento = (PF Não Ajustado +
PF Conversão de Dados) * Fator Ajuste (VAF)

Neste projeto de exemplo não aplicamos conversão de dados e não consideramos o valor de fator de reajuste, dessa forma, o tamanho em pontos de função do aplicativo é de 47.

O que significa esse tamanho?

Esse tamanho por si só não representa muito, mas quando a pergunta correta é feita: "Quanto tempo se leva para desenvolver algo de 47 pontos de função?" aí sim sua importância surge. Esse valor é utilizado como base para os desenvolvedores, que mediante suas experiências anteriores conseguem estimar o esforço para concluir 47 pontos de função. Se os desenvolvedores não possuem essa experiência podemos sugerir um cálculo simples.

Definimos que cada ponto de função equivale à uma hora de trabalho de um desenvolvedor. Assim, um único desenvolvedor, em tese, levaria 47 horas. Mas, nenhum desenvolvedor é 100% produtivo em suas 8 horas diárias de trabalho, afinal de contas, são humanos, não máquinas. Então, digamos que a produção real seja de 6 horas/dia, com uma divisão simples de 47/6 chegamos ao resultado de 7,8 dias que arredondamos para 8 dias de trabalho.

Agora que temos um valor palpável de tempo, com base nele podemos gerenciar custo e prazo.

Conclusão

Ao se falar de qualidade de código é importante ressaltar que existem várias outras métricas, existem até mesmo ferramentas que avaliam e analisam essa qualidade baseados em regras pré-definidas. Um exemplo de tal ferramenta é a FxCop da Microsoft, que avalia código .NET quanto a melhorias de design, desempenho, boas práticas e segurança. Já ao se falar de métricas para estimativas, a APF se mostra muito útil para dar um norte aos desenvolvedores. Pois ao identificarem o tamanho de um projeto, podem realmente estimar o tempo necessário. Contudo, essa estimativa deve levar em conta experiências anteriores, sempre que possível, além de também respeitar o desempenho individual, pois ninguém produz o mesmo que outro.

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi e editor técnico .Net Magazine. Formado em processamento de dados pela FATEC-TQ.

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

DevMedia

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

20 mil posts

430 mil cadastrados

10 milhões de page-views por mês

http://www.devmedia.com.br/

DevMedia